Final Year Project

Intrinsically Motivated Learning and Criticality

Candidate No: 167184

Supervisor: Prof. Luc Berthouze

University of Sussex, Department of Informatics
Bachelor of Science in Computer Science and Artificial Intelligence

2020

Statement of originality

This report is submitted as part requirement for the degree of Bachelor of Science in Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Acknowledgements

Immense thank you to the author's technical supervisor, Prof. Luc Berthouze, for the inspiration for the research question, for his excellent guidance and the generous support that was always given in time, and for the encouragement to get into contact with the authors of the academic work used in this project.

Thank you to Miguel Aguilera for kindly making available the results source code from his paper which was used in the research, as well as for readily answering queries related to changes in the code for the purpose of this work. The author would also like to thank Makoto Otsuka for referencing his PhD thesis in his communication. The detail in the thesis helped significantly during the implementation of a similar architecture from a paper written by him, which was originally intended to be used in this project.

Summary

The investigation of intrinsic motivation in reinforcement learning is key in the path for building truly autonomous systems. Research suggests that adaptivity is the result of maximising a prediction capacity. A system whose goal is to make better predictions about its environment, covers more of its sensorimotor space and acquires more diverse knowledge about the generation of patterns. This behaviour makes the system more adaptive to environmental changes than a system learning to achieve a single environmental state.

Complexity research proposes that adaptive dynamic systems owe their high computational capacity and their autonomous behavior to self-organising critical states. At criticality, the systems are in transition between qualitatively different states and their order parameters vary continuously. It has been shown that during this variability, the systems are very sensitive to external influence and information propagates better between different parts of the system. The constant transitioning between two regimes of behaviour allows for moments of looser local connectivity, while preserving overall stability.

Both intrinsically motivated reinforcement learning and the method of tuning networks to criticality aim to achieve autonomous, adaptive behaviour. However, the interaction of the two has not been explored. This work investigates the influence of criticality on the predictor and the actor of an intrinsically motivated system. Its intrinsic goal is to maximise prediction capacity. The implemented approach to poise the predictor at criticality is agnostic for its training and consists in finding the parameter values which define its critical state. It involves the estimation of a statistical measure which is directly related to the rate of change of the order parameters of a dynamic system. The method for the critical actor uses the distribution of correlations in a physics model whose critical point is known. The actor is instantiated using the reference distribution and is trained to maintain it, while being driven by input.

The environment where the system's behaviour is tested with, and without, the presence of criticality, has states that can be reached via the creation of multiple different sensorimotor patterns. The task has a significant time-lag and requires good memory of the past. Coverage of more states and the execution of more diverse motor patterns under criticality would be evidence for its positive influence.

The results do not show significant differences between the critical and the non-critical models of the learning architecture, however, this might be dependent on the particular methods and on the implementation of the experiments. The research work could be the base for further investigation into the benefits of criticality for the generation of autonomous behaviour.

Table of Contents

1 Introduction

Exploration is a process which is exhibited throughout all stages of a human's life. During childhood, a baby interacts with its body and with objects in its environment in a play-like manner. During adulthood, a person is often looking for ways for solving a problem. The reason behind one's actions in the latter example are fairly clear – there is a goal in mind which needs to be achieved and this is often done so by exploring a space of solutions. Regarding the former, biology and developmental psychology explain this as a necessary learning process for the agent: in order to survive in an environment, an agent needs to have an understanding of the environment's dynamics and how they respond to one's actions (Runco and Jaeger, 2012). They are exploring the world in order to later function in this world.

Despite the fact that adults are driven to explore often because of an externally supplied reward, children are regarded as purely intrinsically motivated. They perform actions done for their own sake. In computational theory exploratory behaviour is a synonym for intrinsically motivated behaviour. An analog for intrinsic motivation as described by mental development theory has been naturally incorporated in reinforcement learning. While in classical reinforcement learning the environment which provides the reward to the learning system is external to it, this environment can also be regarded as part of the learning system (Sutton and Barto, 2018) .Thus, the supplied reward becomes intrinsic. This shift in modelling learning has led to the training of better, adaptive learners whose skills and knowledge are not tied to a specific task or a specific environment but are rather transferrable (Singh *et al.*, 2010).

Dynamical systems provide another theory for the processes governing adaptation to changing conditions. A phenomenon initially observed in non-biological systems, criticality has been hypothesised to govern flexible behaviour in a variety of biological systems (Mora and Bialek, 2011). It has been observed that these systems function between two phases with different dynamical properties – order and disorder. The boundary between the two regimes of behaviour is called the critical point. The critical-brain hypothesis states that the brain is also poised at criticality (Chialvo, 2018) – its complexity arises from the process of switching between two extremes. One is too much order where there are strong local interactions between neurons but no communication between different parts of the brain. The other is too much disorder where the transmission of information is chaotic and ineffective. Criticality has been shown to lead to high sensitivity to external stimuli and to the organisation of locally stable states (Tkacik *et al.*, 2009). These properties might be the reason behind the brain's ability to quickly adapt to different conditions while being robust.

A critical system can be seen as a distributed information-processing system, thus it is naturally relevant for computational theory. A system at the border between a regime where errors propagate endlessly and a regime where changes rapidly vanish might be what constitutes the best tradeoff between robustness and flexibility. This project explores the effect of criticality on an intrinsically motivated system: whether its exploratory behaviour becomes enhanced when it is poised at criticality.

1.1 Literature Review

1.1.1 Intrinsic Motivation

Out of the various approaches to intrinsic motivation in reinforcement learning (Oudeyer, 2007), there are two main types. According to the knowledge-based approach, an agent is maximising its prediction performance. The performance of a predictor is the absolute value of the difference between the prediction of a state/action/reward and the actual value. These prediction errors might be stored on the entire history or part of the history (Schmidhuber, 2010). The difference between the agent's capacity to predict before and after being trained on the last time step's data is what constitutes the intrinsic reward.

In one of the most complete formalisations of knowledge-based theories of curiosity (Schmidhuber, 2010), the prediction capacity is described as compression performance. The smaller the performance, the more regularity in the observations so far. The curious system consists of a predictor of the world whose improvement is what generates the intrinsic reward and a controller which maximises the reward. The level of learning progress can be made higher by either improving the prediction capability of the model or by improving the controller. There are various ways of implementing the two. In (Schmidhuber, 2015), the idea of a controller having the model as its input has been established for the first time. This approach has been used in (Schmidhuber, 2018) although without introducing exploration. In (Srivastava, 2013) the controller and model constitute one single net which self-generates compression and non-compression tasks.

A knowledge-maximising agent is essentially exploring the state space of the environment. Learning certain behaviours that lead to expected outcomes results in the acquisition of skills (Baldassarre, 2013). In contrast, the competence-based approach states that the agent is intrinsically motivated to acquire skills and this is what leads to accumulation of knowledge about the world, not the other way around. It is argued that it is control of the world which is at the core of exploration, not knowledge of the world, thus the agent is discovering the action space instead of the sensorimotor space. Competence-based intrinsic motivation research has gone in the direction of self-generation of goals. In (Santucci, 2016) goal-discovery and goal-selection are performed by autonomously detecting changes in the environment in a robot's visual system and then learning to reach these goals. In (Mannella, 2018) the goal-discovery is pointed at the robot's own sensory changes. In the first model the intrinsic motivation signal is not tied to the formation of goals but only to the selection while in the second model it affects the updates of the representation of goals.

It has been shown that the competence-based approach is more suited for making machines acquire skills faster (Santucci, 2013). However, the knowledge-based approach can be more useful for modelling exploratory behaviour. It does not limit the agent to explore a limited action or task space but to gain knowledge about the whole spectrum of sensorimotor data. Exploring the different ways of generating the same outcome in the environment doesn't make an agent less curious. Purely intrinsically motivated learning as defined in (Schmidhuber, 2010) is not about rapid acquisition of skills but about novel pattern detection in the environment measured by the compression progress due to learning.

1.1.2 Criticality

There is a fairly novel idea for intrinsic motivation coming from the study of criticality in conceptual models. Some researchers think that intrinsic motivation can be found in the agent's exploration of their criticality regime (Aguilera, 2018a). The hypothesis is that while exploring its criticality regime by preserving an internal network of relations (Aguilera, 2018b) or by maximizing the interaction between the components of its neural controller, its sensory input, and its motor behavior, a system seems to solve a task it was not programmed to solve. The researchers describe their approach as agnostic in terms of inputs or an external world. Criticality's effects are being explored on their own.

This differs from the main approach to studying criticality. Most methods focus on understanding criticality in terms of learning to successfully represent an external world. It has been suggested that criticality is a by-product of the process of a predictive system building a model of its environment if this environment is complex enough (Friston, 2012). This resonates with the idea coming from biology that criticality is a revolutionary solution reached by living systems in their constant coping with complex environments (Hidalgo, 2014). Some studies focus on the pure computational abilities of networks operating at the edge of chaos supporting the idea that dynamical systems reach its maximum information processing capacity at the critical point (Goudarzi, 2012). Although it is highly speculated that criticality is entangled with a variety of adaptive processes, it is not clear what exactly the relationship is. Studying criticality as a correlate of other cognitive phenomena like perception and prediction prevents from understanding what the contribution of criticality is and whether it causes adaptivity or is a consequential property of an adaptive system. Moreover, learning about the information processing ability of a critical system does not provide enough ground for explaining robustness and flexibility exhibited at the critical point.

1.2 Problem Specification

Systems which exhibit highly adaptive skills learn about their environments autonomously. A theory for how they do so involves the maximisation of a capacity to predict states and actions in an environment. This way of exploring the environment could lead to faster acquisition of knowledge and skills, and the development of flexibility. A different theory suggests that autonomous behaviour consists in self-organisational activities and the maintenance of specific internal configurations. The possession of these internal constraints might lead to high receptivity to external disturbances, positing the system at the optimum levels of robustness and flexibility.

The work presented in this report combines these ideas for intrinsic motivation in order to investigate the influence of self-generated behaviour on the exploratory abilities of the system. Critical properties are infused in a curious system which is constantly improving its prediction capacity, in order to investigate whether criticality helps it learn better in its environment.

The explorative system, whose controller maximises the performance of its predictor, is tested for learning improvement, with and without criticality present in the controller or in the predictor. It is shown whether the critical system covers more states or whether it learns all possible ways of reaching a particular state before moving on to a different one. It is investigated how much the predictor improves its prediction capacity under criticality, and how much criticality accelerates the maximisation of the received rewards, both internal and external.

1.2.1 The task

In order to test these research hypotheses, the system is embodied in an environment, comprised of a variety of states with different levels of complexity. It is deterministic but the state space allows for multiple ways of reaching complex states. The system explores the environment by exploring its sensorimotor space. It learns about the outcomes of different states and actions by predicting them and improving after receiving prediction errors. Each state can be achieved by performing a specific sequence of actions. More difficult to reach states involve more actions. The agent achieves simple states first and gradually improves on what it's learned by reaching more complex states. The knowledge needed for successful exploration/exploitation of the environment consists in better coverage of states, improvement in prediction capacity and better achievement of external goals.

1.3 Report Structure

The report is organised as follows. Section 2 explains how the project complies with the British Computing Society's standards. Section 3 describes separately each element of the model architecture: the intrinsic motivation framework, the two network models, their learning methods and the methods to poise them at criticality, and goes on to explain how they form the project's complete model architecture together. Section 4 explains how the elements were implemented and what approaches were taken in order to assess their performance. Section 5 describes the task for which the models were trained, together with the models' parameter configurations. Section 6 provides the results of the experiments. Section 7 presents a discussion on the results and describes the directions future work could go into.

2 Professional Considerations

This project adhered to the professional considerations outlined in the BCS Code of Conduct (BCS, 2015) in its entire continuity. The details of the precise points followed in each section of the Code are listed below.

2.1 Public Interest

Since this is a research-based project which has not made use of public data, nor has required the participation of others, points 1.a and 1.c of the BCS Code of Conduct which demand that the author shall respect "the privacy, security and wellbeing of others and the environment" and not discriminate others on any social or personal grounds are of no relevance. The author considered point 1.d which demands that this work should seek to promote the inclusion of all sectors in society and the benefits of IT, but the software used in the project was targeted to be used only by the author, so no opportunity to do the above arised. The project adheres to p. 1.b since all the used work and advice by Third Parties is well documented, ensuring that their legitimate right to be referenced is respected.

2.2 Professional Competence and Integrity

In compliance with the BCS Code of Conduct 2.a and 2.b, part of the research area of this work is one that the author has some competence in, having studied the Introduction to Machine Learning and Neural Networks modules at the University of Sussex. Having attended the Introduction to Complex Systems summer school in the University of Utrecht, the research field of complexity is one of which the author can claim a ground-level understanding and which is enough to be the basis for further self-study and development. Constantly improving their knowledge and competence is in accordance with point 2.c. which demands that these are being developed on a continuing basis. The author has considered and followed the BSC Code of Conduct rigorously which ensures that they have the necessary knowledge of the Legislation and they carry out their professional responsibilities with care, following p. 2.d.

All significant information taken from Third Parties and used in the project, as well as claims and views of Third Parties presented in the project, especially these which are part of the research area where the author has no significant experience in, is referenced and well documented. This ensures that no harm is done to the work of others or to their reputation, in compliance with point 2.f.
Apart from communication with their technical supervisor, the author has not sought honest criticisms directly related to their work, but has discussed ideas related to their work in their communication with some Third Parties, therefore adhering to p. 2.e of the Code which states that alternative viewpoints shall be valued. No offer of unethical inducement has been made, complying with point 2.g.

2.3 Duty to Relevant Authority

The project was carried out with due care and diligence with accordance with the University of Sussex requirements and no conflict of interest between the author and the University has been given rise to, complying with points 3.a and 3.b of the Code. The author completely accepts responsibility for their work and ensures that no confidential information was disclosed, in accordance with points 3.c and 3.d. No information about the software used in the project was misrepresented and no advantage of the lack of relevant knowledge of others was taken, complying with point 3.e.

2.4 Duty to the Profession

In accordance with p. 4.a and 4.c., the author accepts their personal duty to "uphold the reputation of the profession" and of the BCS. Where shortcomings of used existing software were found, they were respectfully communicated to the relevant Third Parties to provide them with encouragement and support of their professional development, in doing so complying with the BCS Code of Conduct 4.d and 4.f Where the author taught that certain methods in the academic work used in this project were presented in an unclear and/or imprecise way, they kindly sought explanation from the authors in accordance with the latter points, as well as with the Code's 4.b, with the hope to improve professional standards if enough ground for such improvement was present.

3 Methodology

In order to test the influence of criticality on the performance of an intrinsically motivated learning system, the model architecture consists of two entities: the first represents the intrinsic motivation and the memory of the system, and the second represents the actor of the system.

The motivator of the system, here called the predictor, is implemented as an Echo-State Network, and the actor is implemented as a Restricted Boltzmann Machine (Otsuka, 2010; Otsuka *et al.*, 2011). The two are connected directly via a memory signal and an intrinsic reward signal (Fig. 1). The ESN is tuned to criticality by finding the critical region of its hyper-parameter space (Livi, Bianchi and Alippi, 2018). The method is robust to different network configurations and is independent from the training procedure. The method to turn the RBM critical involves a learning rule: the weights and the biases of the network are being adjusted to preserve their correlations the same as a model universality class which is at its critical point (Aguilera and Bedia, 2018).

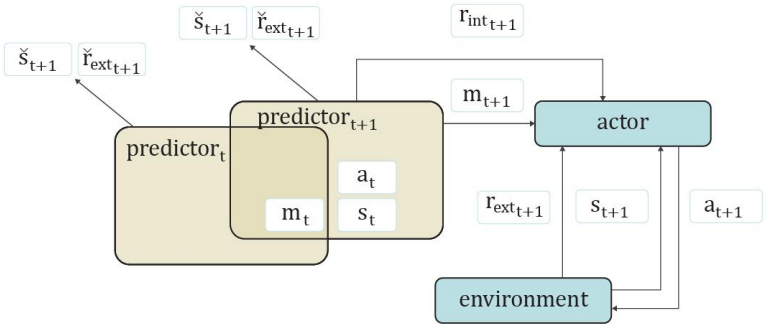


Fig. 1: Schematic view of the actor-predictor architecture. Schematic view of the actor-predictor architecture. The actor receives state and external reward from the environment, as well as an internal reward signal from the predictor. It uses the combination of the rewards for learning. It uses the predictor's state as memory in order to choose the next action to perform.

The model architecture consists of two networks and three learning methods, since the learning procedure is what defines the critical actor. Both the actor and the predictor are tested with and without the presence of criticality, resulting in a two by two experimental design.

This section will present each element of the model architecture separately, by describing the adopted frameworks and learning methods, and will show how they are combined to create the joint architecture.

3.1 Intrinsic Reward

3.1.1 Reinforcement Learning

Reinforcement learning is a sub-field of machine learning which has the aim of learning the optimal behaviour in a given environment towards a given goal. It is different from unsupervised and supervised machine learning because it focuses on the direct interaction with an environment (Sutton and Barto, 2018). The system learns by sampling the state and the action space of the environment by trial and error, possessing no prior knowledge of the dynamics of the environment or its reward function. It aims to maximise a collected reward which is comprised of an immediate reward and a distant reward.

Tasks in reinforcement learning are formalised according to the framework of Markov decision processes. In its simplest form, a Markov decision process (MDP) consists of a state space, an action space and an expected reward function which can be defined as a state-action pair. In a discrete task, at each time step t, the agent performs an action and receives a tuple of two elements from the environment: the next state it lands at following the state-transition probabilities and the immediate reward which is defined by the reward function. A state is considered to be a proper Markov state if it contains all aspects of the past interaction between the agent and the environment, which are necessary to make a decision for the future.

The different reinforcement learning algorithms consist in the estimation of state-value functions or action-values functions – determining how good it is to be in a given state or to perform a given action in a given state. They are learned iteratively, by sampling the environment many times. These functions are determined by policies – mappings between states to probabilities of selecting each action. At each time step t, an agent chooses an action under a certain policy it is following or by estimating the value of the state (or state-action pair) it is currently in. Methods where both the probabilities of selecting actions in a state and the value functions are estimated are called actor-critic methods: the actor learns a policy by making use of the value estimate coming from the critic. The critic can apply the value function on the first, as well as on the second state transition.

3.1.2 Intrinsic Motivation

The boundary between agent and environment in reinforcement learning does not necessarily represent the physical boundary of a body. It can be drawn between the agent and its sensors and motors, or it can represent internal processes present in the agent (Error! Reference source not found.). The general rule is that everything outside of the agent's absolute control is part of the environment (Sutton and Barto, 2018). The reward computation must therefore be external to the agent, so that it cannot modify it arbitrarily. Actor-critic methods can be used to make a differentiation between an internal reward coming from the "body", or the internal part of the environment, and a reward coming from the external environment. The internal reward can represent the improvement of the ability of the critic to estimate value functions. In this way, the goal of the actor is directed inwardly, yet it cannot be directly changed, and it is related to external states.

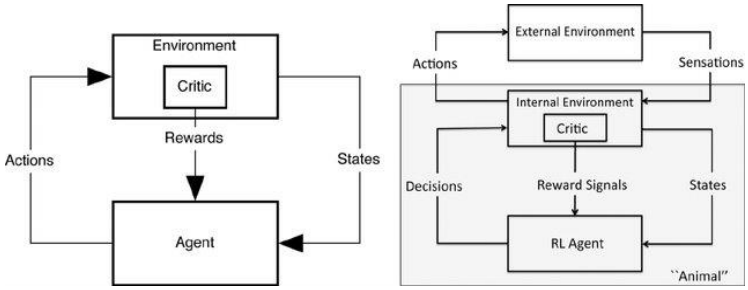


Figure 2: An agent interacting with an external environment and an agent interacting with both an internal and an external environment, (Singh et al., 2010)

It is important to make a distinction between methods which use internal rewards in order to accelerate the maximisation of an expected external reward and those which use such a reward in order to represent an intrinsic goal (Schmidhuber, 2010). An intrinsic goal is to actively and autonomously generate and learn patterns in an environment. The task of such agents is to execute actions which lead to the generation of new, learnable environmental regularities. Similarly to a classic actor-critic method, the knowledge-based framework for maximising creativity (Schmidhuber, 2010) involves two learning entities – a controller and a predictor. The controller learns to perform better actions by estimating value functions and the predictor learns to predict environmental states. In a purely intrinsically motivated system, the reward which the controller maximises is the improvement of the predictor. In the case of estimating state-action values, the values represent the degree to which the generated actions lead to learnable state transition probabilities. Formally, the predictor's ability is defined as

$$\boldsymbol{C_{xry(p,h\leq t)}} = \sum_{\tau=1}^{t} \left|\boldsymbol{pred}\left(\boldsymbol{p}, \boldsymbol{x}(\tau) - \boldsymbol{x}(\tau)\right)\right|^2 + \left|\boldsymbol{pred}\left(\boldsymbol{p}, \boldsymbol{r}(\tau)\right) - \boldsymbol{r}(\tau)\right|^2 + \left|\boldsymbol{pred}\left(\boldsymbol{p}, \boldsymbol{r}(\tau)\right) - \boldsymbol{y}(\tau)\right|^2 \quad (3.1.1)$$

which is the sum of the prediction errors for the state, action and reward for the entire history of the agent, and the predictor improvement is defined as

$$\boldsymbol{r_{int}}(\tau) = \boldsymbol{f}(\boldsymbol{C}(\boldsymbol{p_{old}}, \boldsymbol{h_{old}}), \boldsymbol{C}(\boldsymbol{p_{new}}, \boldsymbol{h_{old}})) \quad (3.1.2)$$

which is the difference between the predictor's quality before and after having learnt from the last history.

3.1.3 Creativity Framework

This project has reformulated the asynchronous creativity maximisation framework (Algorithm 1) to work synchronously as part of the actor-predictor reinforcement learning system (Algorithm 2). While in the original framework the predictor steps 2,4 and 5 are defined to cover an arbitrary number of time steps, in the modified version, used in this project, the steps are performed at the current time step $\boldsymbol{t}$. In this way, the actor receives an internal reward at each time step, meaning that it is constantly using its intrinsic motivation to learn.

Algorithm 1 - Asynchronous framework for maximising creativity (Schmidhuber, 2010)

Define $\boldsymbol{p(t)}$ as the agent's current predictor at time $\boldsymbol{t}$
Define $\mathbf{c(t)}$ as the agent's current controller

Controller: For each $\boldsymbol{t}$ $(\mathbf{1} \leq \boldsymbol{t} < \boldsymbol{T})$ do:

1) Let $\mathbf{c(t)}$ execute $\boldsymbol{a(t)}$.
2) Collect $\boldsymbol{s(t+1)}$.
3) Check if there is non-zero creativity reward $\mathbf{r_{int}(t+1)}$. If not, set $\mathbf{r_{int}(t+1)} = \mathbf{0}$.
4) Create $\mathbf{c(t+1)}$ by improving $\mathbf{c(t)}$.

Predictor: Set $\mathbf{p_{new}}$ to the initial data predictor and repeat until $\boldsymbol{T}$:

1) Set $\boldsymbol{p_{old}} = \boldsymbol{p_{new}}$; Observe time step $\boldsymbol{t}$ and set $\boldsymbol{h_{old}} = \boldsymbol{h}(\leq \boldsymbol{t})$.
2) Evaluate $\boldsymbol{p_{old}}$ on $\boldsymbol{h_{old}}$ acc. to (1) in order to obtain $\boldsymbol{c_{old}}$. This can take many time steps.
3) Use $\mathbf{h_{old}}$ to obtain a better predictor $\boldsymbol{p_{new}}$.
4) Evaluate $\boldsymbol{p_{new}}$ on $\boldsymbol{h_{old}}$ in order to obtain $\boldsymbol{c_{new}}$. This can take many time steps.
5) Generate creativity reward for time step τ acc. to (2).

Algorithm 2 – Synchronous modification of the maximising creativity framework

Define $\boldsymbol{p(t)}$ as the agent's current predictor at time $\boldsymbol{t}$
Define $\mathbf{c(t)}$ as the agent's current controller at time $\boldsymbol{t}$
For each $\boldsymbol{t}$ $(\mathbf{1} \leq \boldsymbol{t} < \boldsymbol{T})$ do:

1) Let $\mathbf{c(t)}$ execute $\boldsymbol{a(t)}$.
2) Collect $\mathbf{h(t+1)}$.
3) Evaluate $\mathbf{p(t)}$ on $\mathbf{h(\leq t)} = \boldsymbol{c_{old}}$.
4) Generate $\mathbf{p(t+1)}$ by using $\mathbf{h(t)}$.
5) Evaluate $\mathbf{p(t+1)}$ on $\mathbf{h(\leq t)} = \boldsymbol{c_{new}}$. Set $\mathbf{r_{int}} = \mathbf{c_{new}} - \mathbf{c_{old}}$, if $\mathbf{r_{int}} > \mathbf{0}$, otherwise set $\boldsymbol{r_{int}} = \mathbf{0}$.
6) Create $\mathbf{c(t+1)}$ by using $\mathbf{r_{int}}$ to improve $\mathbf{c(t)}$.

The learning methods for the predictor and the controller which constitute steps 4 and 6 are described in the following sections.

3.2 Echo-State Network

The echo-state approach in machine learning consists in the use of a recurrent neural network as a reservoir which is randomly generated and whose weights are not trained. Only the readout of the reservoir is trained which makes the training conceptually simple and computationally inexpensive, unlike traditional RNN training. Echo-state networks preserve the most important characteristics of RNNs – a large memory and very adaptable computational abilities (Lukoševičius, 2012), which makes them good for supervised temporal ML tasks.

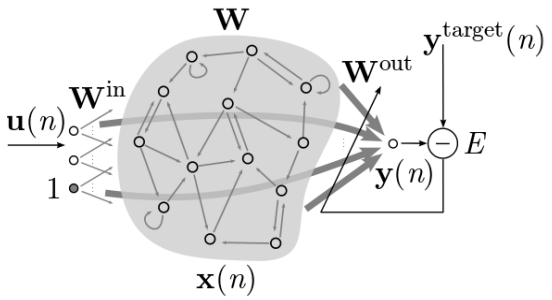


Figure 3: An echo-state network (Lukoševičius, 2012)

The training of the echo-state network consists in feeding it with an input signal $\boldsymbol{u(n)} \in \mathbb{R}^{\boldsymbol{N_u}}$ and a target output signal $\boldsymbol{y^{target}(n)} \in \mathbb{R}^{\boldsymbol{N_y}}$ which is known, with $\boldsymbol{n = 1, \ldots, T}$, each time step in the sequence, and $\boldsymbol{T}$ is the length of the sequence. The task of the ESN is to learn a model with output $\boldsymbol{y(n)}$ which is as close as possible to the target output, minimising an error measure $\boldsymbol{E(y, y^{target})}$, and to also generalise well to new, unseen data. The error measure is usually a Mean-Square Error:

$$\boldsymbol{E(y, y^{target})} = \frac{\boldsymbol{1}}{\boldsymbol{N_y}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{N_y}} \sqrt{\frac{\boldsymbol{1}}{\boldsymbol{T}} \sum_{\boldsymbol{n=1}}^{\boldsymbol{T}} \left(\boldsymbol{y_i(n) - y_i^{target}(n)} \right)^{\boldsymbol{2}}} \tag{3.2.1}$$

The update equation for the non-leaky units of the reservoir of the network without output feedback is

$$\boldsymbol{x(n+1) = f\left(W^{in}[1; u(n+1)] + W x(n) \right)}, \tag{3.2.2}$$

where $\boldsymbol{x(n)}$ is a vector of reservoir unit activations, 1 is the bias unit for the input layer which is vertically concatenated to it, $\boldsymbol{W^{in}}$ is the input weight matrix and $\boldsymbol{W}$ is the matrix for the recurrent weights in the reservoir. $\boldsymbol{f}$ is usually the hyperbolic tangent or sigmoid function. The readout layer is defined as

$$\boldsymbol{y(n+1) = f^{out}(W^{out}[1; u(n+1); x(n+1)])} \tag{3.2.3}$$

Here the output weight matrix $\boldsymbol{W^{out}}$ is multiplied by the concatenated vector of the input layer bias, the input and the collected reservoir activations vector, and $\boldsymbol{f^{out}}$ is usually the hyperbolic tangent function.

3.2.1 Off-line training

The procedure to train the ESN offline (Lukoševičius, 2012) is:

1) Randomly initialise the reservoir weights matrix $\boldsymbol{W^{in}}$ using the following hyper-parameters: sparsity, spectral radius and number of reservoir units.
2) Randomly initialise the input weights matrix $\boldsymbol{W^{in}}$ defined by the input weights scaling parameter $\boldsymbol{a}$.
3) Run the network with the input signal, starting from an arbitrary reservoir state $\boldsymbol{x(0)}$ and updating the reservoir state for each input instance according to eq. (3.2.2).
4) Use linear regression to compute the linear output weights $\boldsymbol{W^{out}}$ minimising the MSE between the network output $\boldsymbol{y(n)}$ and the target output $\boldsymbol{y^{target}(n)}$.
5) Use the trained network on new input data $\boldsymbol{u(n)}$ by updating its reservoir states and using the trained output weights $\boldsymbol{W^{out}}$ to compute the output $\boldsymbol{y(n)}$.

3.2.2 On-line training

The online version of the training procedure involves the use of the recursive least squares (RLS) algorithm known from linear signal processing (Jaeger, 2003). In order to compute a weight vector $\boldsymbol{W^{out}}$ at each time step which will determine a good model of the current training input-output, the RLS algorithm minimises the error

$$\sum_{\boldsymbol{k=1}}^{\boldsymbol{n}} \boldsymbol{\lambda^{n-k}} \left((\boldsymbol{f^{out}})^{-1} \boldsymbol{y_{teach}(k)} - (\boldsymbol{f^{out}})^{-1} \boldsymbol{y_{[n]}(k)} \right)^{\boldsymbol{2}} \tag{3.2.4}$$

where lambda λ is the forgetting factor and $\boldsymbol{y_{[n]}(k)}$ is the network output which would be obtained at time $\boldsymbol{k}$ if the current output weights $\boldsymbol{W_n^{out}}$ were employed at all times $\boldsymbol{k = 1, \ldots, n}$. The input to the RLS algorithm is the reservoir activations vector $\boldsymbol{x_n}$ collected according to (3.2.2) and the output to be estimated is the teacher signal $\boldsymbol{y(n)}$.

The algorithm for uncentered data (Lewis, 2020b) is the following:

1. Record $\boldsymbol{u(n)}$ and $\boldsymbol{y_{teach}(n)}$ from the input signal $\boldsymbol{U(n)}$.
2. Calculate the inverse sample covariance matrix $\boldsymbol{P(n)}$ from $\boldsymbol{P(n-1)}$ and $\boldsymbol{u(n)}$.
3. Calculate $\boldsymbol{W^{out}(n)}$ from $\boldsymbol{W^{out}(n-1)}$, $\boldsymbol{u(n)}$, $\boldsymbol{y_{teach}(n)}$ and $\boldsymbol{P(n)}$.

The update equation for the weights at each time step t is:

$$\boldsymbol{W^{out}(k) = W^{out}(k-1) + P(k)u(k)^T [y(k) - u(k)W^{out}(k-1)]} \tag{3.2.5}$$

The update for $\boldsymbol{P(k)}$ uses the "matrix inversion lemma" (Lewis, 2020b) and results in

$$\begin{aligned} \boldsymbol{P(k)} &= \left(\boldsymbol{P(k-1) + u(k)^T u(k)} \right)^{\boldsymbol{-1}} \\ &= \boldsymbol{P(k-1) - \frac{P(k-1)u(k)^T u(k) P(k-1)}{1 + u(k)P(k-1)u(k)^T}} \end{aligned} \tag{3.2.6}$$

3.3 Restricted Boltzmann Machine

A Boltzmann Machine is an undirected graphical model which consists of two layers of binary nodes: a visible layer $\boldsymbol{V} = \{\boldsymbol{V_1}, \dots, \boldsymbol{V_N}\}$, and a hidden layer $\boldsymbol{H} = \{\boldsymbol{H_1}, \dots, \boldsymbol{H_K}\}$. In the fully-connected model, each node $\boldsymbol{i}$ is connected to each node $\boldsymbol{j}$ by a symmetric weight $\boldsymbol{w_{ij}}$. In the restricted model (Fig. 4), weights occur only between $\boldsymbol{V}$ and $\boldsymbol{H}$. The biases of the visible nodes are $\boldsymbol{b^v}$ and the biases of the hidden nodes are $\boldsymbol{b^h}$. The stochastic behaviour of the restricted Boltzmann Machine for a particular configuration is characterized by the energy:

$$\boldsymbol{E(v, h; \theta)} = \boldsymbol{-v^T W^{vh} h - v^T b^v - h^T b^h} \tag{3.3.1}$$

The restricted model ensures that the hidden units are decoupled when visible units are clamped to their observed values (Otsuka, 2010) and it allows for an easy computation of the posterior distribution of the hidden layer.

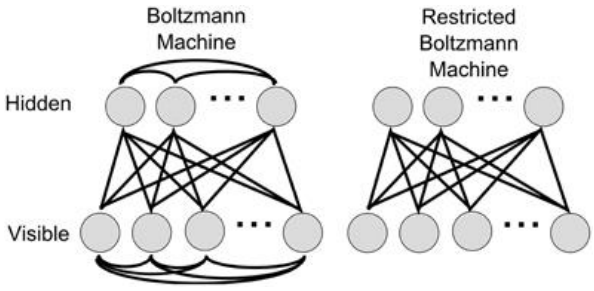


Fig. 4: The Boltzmann machine: the fully-connected model has weights within the hidden and visible layers, while the restricted model has weights only between the layers. (O'Connor et al., 2013)

The equilibrium free energy of the visible layer $\boldsymbol{F(v)}$ can be expressed as the expected energy $\boldsymbol{E}$ minus an entropy (Sallans and Hinton, 2004):

$$\boldsymbol{F(v)} = \sum_{\boldsymbol{h}} \boldsymbol{P(h|v) E(v, h)} + \sum_{\boldsymbol{h}} \boldsymbol{P(h|v) \log P(h|v)} \tag{3.3.2}$$

where $\boldsymbol{P(h|v)}$ is the Boltzmann distribution which achieves optimal trade-off between the two terms: it makes the first term low by putting a lot of mass on the hidden configurations for which the expected energy $\boldsymbol{E(v, h)}$ is low, and makes the second term low by having a high-entropy.

3.3.1 Free-Energy and Reinforcement Learning

The RBM can be used for reinforcement learning by dividing the visible layer into a state and an action layer (Fig. 5).

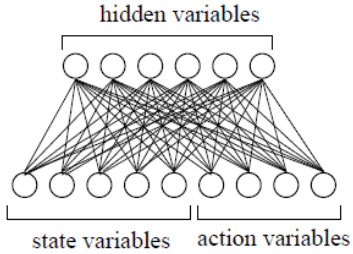


Fig. 5: The visible layer of the RBM is split into state and action variables.

The equilibrium free energy of a state-action pair is then defined as:

$$\boldsymbol{F(s,a;\,\theta) = -\sum_{i,k} v_i W_{ik}^{vh} \hat{h}_k - \sum_i v_i b_i^v - \sum_k \hat{h}_k b_k^h} \tag{3.3.3}$$

$$\boldsymbol{+ \sum_k \hat{h} \log \hat{h}_k + \sum_k (1-\hat{h}_k) \log(1-\hat{h}_k)}$$

Where $\boldsymbol{\hat{h}_k = P(h_k = 1 | s_t, a_t) = \left(1 + \exp\left(-\sum_{i=1}^{|V|} w_k^{vh} v_{t,i} - b_k\right)\right)^{-1}}$ is the expected value of each hidden unit. For a time-step $\boldsymbol{t}$, $\boldsymbol{\hat{h}_t}$ is computed by clamping the state nodes $\boldsymbol{S = s_t}$ and the action nodes $\boldsymbol{A = a_t}$.

The negative free energy is used to approximate the Q-value of a state-action pair and to form a modified SARSA temporal-difference update rule:

$$\boldsymbol{\delta_t(\theta)^2 \triangleq r_{t+1} - \gamma F(s_{t+1}, a_{t+1}; \theta) + F(s_t, a_t; \theta)} \tag{3.3.4}$$

The episodic update rule of the parameters is given as:

$$\boldsymbol{\Delta\theta = -\alpha \nabla_\theta J_T^{TD}} \tag{3.3.5}$$

Where $\boldsymbol{\alpha}$ is a learning rate and $\boldsymbol{J^{TD}}$ is the average square TD error to be minimised as:

$$\boldsymbol{\nabla_\theta J_T^{TD} = \frac{1}{T} \sum_{t=0}^{T} \delta_t \nabla_\theta F(s_t; a_t; \theta)} \tag{3.3.6}$$

The derivatives of the free energy with respect to each parameter of the network are expressed as:

$$\boldsymbol{\frac{\partial F(s,a,\theta)}{\partial w_{ik}^{vh}} = -v_i \hat{h}_k} \tag{3.3.7}$$

$$\boldsymbol{\frac{\partial F(s,a,\theta)}{\partial b_j^s} = -s_i}$$

$$\boldsymbol{\frac{\partial F(s,a,\theta)}{\partial b_j^a} = -a_j}$$

$$\boldsymbol{\frac{\partial F(s,a,\theta)}{\partial b_k^h} = -\hat{h}_k}$$

Actions can be selected using a sampling method or by setting the state nodes $\boldsymbol{S} = \boldsymbol{s_t}$ and calculating the free energy for all possible actions by setting the action nodes $\boldsymbol{A} = \boldsymbol{a_t}$ **for** $\boldsymbol{a} = \boldsymbol{1}, .., \boldsymbol{N_a}$. The softmax action selection rule is used with inverse temperature $\boldsymbol{\beta}$:

$$\boldsymbol{\pi(s, a; \theta, \beta) = p(a|s; \theta, \beta) = \frac{\exp\left(-\beta F(s, a, \theta)\right)}{\sum_{a'} \exp\left(-\beta F(s, a', \theta)\right)}} \tag{3.3.8}$$

3.4 Criticality in Networks

This section describes the way criticality is introduced for the echo-state network and the restricted Boltzmann machine. The method for the former finds the critical configuration set of parameters, while the method for the latter is defined by a learning rule.

3.4.1 Tuning the ESN to Criticality

The method to determine the edge of criticality in the ESN uses a non-parametric estimator of the Fisher information matrix (FIM) (Livi, Bianchi and Alippi, 2018), in order to establish the configuration of hyper-parameters of the network which poises it at its critical point.

A thermodynamic system has order and control parameters. The control parameters of the system determine its phase transitions while the order parameters are used to characterise the different phases. When a controlled thermodynamic system is at its critical point, this means that due to the critical value of the control parameter, the system is undergoing a phase transition. During second-order transitions, its order parameters vary continuously and the system can maintain such a state and operate in it. Since the FIM is directly related to the rate of change of the order parameters with respect to the control parameters, it is maximised during critical transitions.

In the case of the ESN, the control parameters are the hyper-parameters of the network. Maximising the FIM can identify the values of the hyper-parameters, when the network is critical, e.g. when it is the most sensitive to inputs and can generate any complex task-related dynamics. Fig. 6: The approach to identify phase transitions in thermodynamic systemsFig. 6 shows the link between the FIM, thermodynamic systems and echo-state networks.

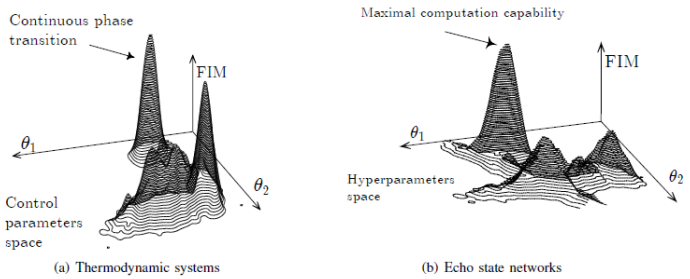


Fig. 6: The approach to identify phase transitions in thermodynamic systems can be adopted to identify the maximum computation capability of the ESN, (Livi, Bianchi and Alippi, 2018).

The FIM is a symmetric positive semidefinite matrix with $\boldsymbol{d(d+1)/2}$ distinct entries which characterise the sensitivity of the PDF $\boldsymbol{p_\theta}$ of the observed data with respect to the $\boldsymbol{d}$-number of hyper-parameters of the network. Each element of the FIM is defined as

$$\boldsymbol{F_{ij}(p_\theta(.)) = \int_{\mathcal{D}} p_\theta(u) \left(\frac{\partial ln p_\theta(u)}{\partial \theta_i}\right) \left(\frac{\partial ln p_\theta(u)}{\partial \theta_j}\right) du} \qquad (3.4.1)$$

where $\boldsymbol{\theta} = [\boldsymbol{\theta_1}, \dots, \boldsymbol{\theta_d}]^{\boldsymbol{T}}$ defines the hyper-parameter space under consideration, $\boldsymbol{p_\theta}(.)$ is the parametric PDF, and $\mathbf{ln}\boldsymbol{p_\theta}(.)$ represents a log-likelihood function.

Since during network training, the PDF of the input often can't be computed or its link to the hyper-parameters is unknown, the FIM can't be directly calculated. Instead, the proposed method estimates the FIM by using a divergence measure between the parametric PDF under consideration and a perturbed version of it $\widehat{\boldsymbol{\theta}} = \boldsymbol{\theta} + \boldsymbol{r}$, with $\boldsymbol{r} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma^2 I_{d \times d}})$ a small normally distributed perturbation vector. The equation for the estimated half-vector representation[1] of the FIM is:

$$\widehat{\boldsymbol{F}}_{\boldsymbol{hvec}} = \left(\boldsymbol{R^T R}\right)^{-1} \boldsymbol{R^T v_\theta} \tag{3.4.2}$$

where $\boldsymbol{v_\theta} = [\boldsymbol{v_\theta}(\boldsymbol{r_1}), \dots, \boldsymbol{v_\theta}(\boldsymbol{r_M})]^{\boldsymbol{T}}$, with $\boldsymbol{v_\theta}(\boldsymbol{r_i}) = \boldsymbol{2D_a}\left(\boldsymbol{p_\theta}, \boldsymbol{p_{\widehat{\theta}_i}}\right), \boldsymbol{i} = \mathbf{1}, \dots, \boldsymbol{M}$ and $\boldsymbol{D_a}(.,.)$ is the result of the Friedman-Rafsky test-statistic[2]. Each input $\boldsymbol{i}$ to the test-statistic are the sets of reservoir activations of a network configured with the parameters to be evaluated $\boldsymbol{\theta}$, and a network configured with the perturbed parameter set $\boldsymbol{\theta_i}$.

$\boldsymbol{R}$ is a $\boldsymbol{M\ x\ d(d+1)/2}$ matrix containing all $\boldsymbol{M}$ direction component pairs of the perturbation vectors, each defined as

$$\left[\boldsymbol{u_{11}^2}, \dots, \boldsymbol{u_{1d}^2}, \boldsymbol{2u_{11}u_{12}}, \dots, \boldsymbol{2u_{1(d-1)}u_{1d}}\right] \tag{3.4.3}$$

The least-squares equation (3.4.2) does not guarantee that the found approximation of the FIM will be a positive semidefinite matrix, therefore its estimation involves the following convex optimisation problem:

$$\begin{aligned} &\boldsymbol{minimise}_{\boldsymbol{F_{hvec}}} && \|\boldsymbol{RF_{hvec}} - \boldsymbol{v_\theta}\|^{\boldsymbol{2}} \\ &\boldsymbol{subject\ to} && \boldsymbol{F_{hvec}(i)} = \boldsymbol{diag}\left(\boldsymbol{mat}\left(\widehat{\boldsymbol{F}}_{\boldsymbol{hvec}}\right)\right), \boldsymbol{i} \in \{\mathbf{1}, \dots, \boldsymbol{d}\}, \\ & && \boldsymbol{mat}(\boldsymbol{F_{hvec}}) \succcurlyeq \mathbf{0}_{\boldsymbol{d \times d}} \end{aligned} \tag{3.4.4}$$

The full algorithm takes a set of parameter configurations, estimates the FIM for each and calculates the matrix's determinant. It returns the parameter set whose FIM has the maximum determinant. A high-level depiction of the procedure is given in *Fig. 7*.

[1] $\hat{F}_{hvec} = \left[\hat{f}_{11}, \dots, \hat{f}_{dd}, \hat{f}_{12}, \dots, \hat{f}_{d(d-1)}\right]^T$ where the diagonal elements are located in the first components of the vector is the half-vector representation of a symmetric matrix.

[2] The Friedman-Rafsky multi-variate two sample test statistic $C(X_p, X_q)$ calculates the number of edges between a data point from p to a data point from q from the concatenated dataset's Euclidean minimal spanning tree (MST)

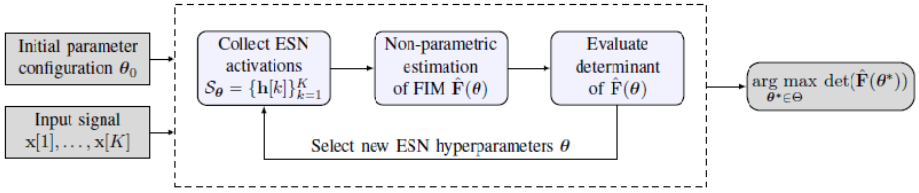


Fig. 7: Schematic, high-level description of the proposed unsupervised learning method (Livi, Bianchi and Alippi, 2018)

The algorithm is agnostic in terms of network configurations and training procedures because it doesn't use the readout layer of the network, but only the reservoir states.

3.4.2 Adapting the RBM to Criticality

In physics, a universality class is a collection of models which share the same scale invariant limit and the same critical exponents. These systems exhibit similar behaviour near these values, independent of their physical properties. In particular, the systems of the same universality class preserve the same distribution of spatial correlations at criticality (Aguilera and Bedia, 2018). This distribution is easily solvable analytically for two-dimensional lattice models.

The Boltzmann Machine is such a model, known as the Ising model in physics. The proposed learning rule which drives it to its critical point maintains the distribution of the correlations between its units the same as the calculated distribution of a model whose critical point is known.

The method uses as reference correlations the correlations of a $\mathbf{20} \times \mathbf{20}$ square lattice Ising model with biases zero and bidimensional weights whose critical point is at $\boldsymbol{W} = \mathbf{log}(\mathbf{1} + \sqrt{\mathbf{2}})/(\mathbf{2}\boldsymbol{\beta})$. The model is simulated using Glauber Dynamics. The distribution of correlations in the system, $\boldsymbol{P}(\boldsymbol{c_{ij}})$, where $\boldsymbol{c_{ij}} = \boldsymbol{s_i s_j}$ is the correlation between units $\boldsymbol{i}$ and $\boldsymbol{j}$, is a power-law distribution (Fig. 8).

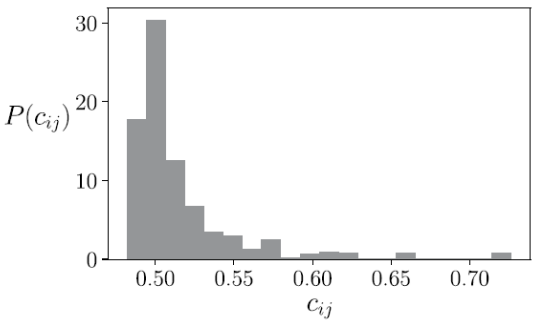


Fig. 8: Distribution of correlation values used for learning (Aguilera and Bedia, 2018)

The new model to be trained is generated with a set of reference correlation values $\boldsymbol{c_{ik}^*}$, $\boldsymbol{k} = \mathbf{1}, \ldots, \boldsymbol{N}$, assigned to each unit, drawn at random from $\boldsymbol{P}(\boldsymbol{c_{ij}})$. At each step the actual correlation values between each unit $\boldsymbol{i}$ and its surrounding units $\boldsymbol{j}$ are calculated as

$\boldsymbol{c_{ij}^{m}}$. The reference values used for learning are generated by sorting the reference values $\boldsymbol{c_{ik}^{*}}$ to match the order of $\boldsymbol{c_{ij}^{m}}$. The ordered values are denoted as $\boldsymbol{c_{ij}^{*}}$. Since the biases of the reference model are zero, the means $\boldsymbol{m_i^{*}}$ are all set to zero. The learning consists in finding which combination of $\boldsymbol{b_i}$ and $\boldsymbol{W_{ij}}$ generates which combination of $\boldsymbol{m_j}$ and $\boldsymbol{c_{ij}}$. This is solved by the following gradient descent rule:

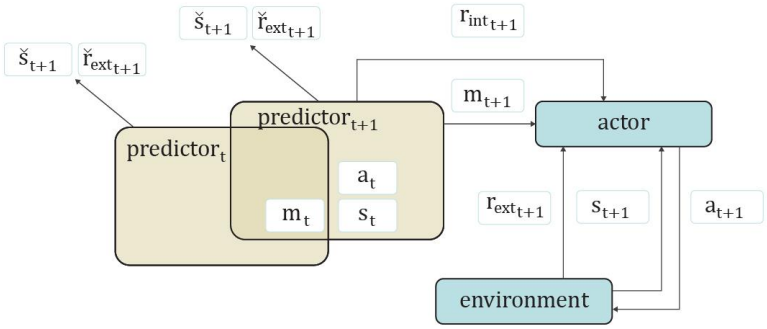
$$\begin{aligned} \boldsymbol{b} &\leftarrow \boldsymbol{b} + \boldsymbol{\mu}(\boldsymbol{m_i^{*}} - \boldsymbol{m_i^{m}}) \\ \boldsymbol{W_{ji}} &\leftarrow \boldsymbol{W_{ji}} + \boldsymbol{\mu}(\boldsymbol{c_{ij}^{*}} - \boldsymbol{c_{ij}^{m}}) \end{aligned} \tag{3.4.5}$$

At each step $\boldsymbol{t, for\ t = 1, \dots, T}$, a sequential update of the units is performed by using Glauber Dynamics.

In the case when the network is embodied, its visible units are split into sensor and motor units. The values of the motor units encode the actions performed by the network. The biases of the sensor units encode the inputs from the environment. At each time-step t, the network's sensors point to $\boldsymbol{s_t}$. In-between the sequential updates of the units, at random, a move is made by the network, in which the action taken $\boldsymbol{a_t}$ is decoded from the current values of the motor units. After the move is made, the network's biases of the sensor units are updated to encode the state $\boldsymbol{s_{t+1}}$, and they maintain their values until the next move, at time $\boldsymbol{t + 1}$.

3.5 Joint Architecture

The author has employed a 2-entity model architecture for solving Partially Observable Markovian Decision Processes (Otsuka, 2010; Otsuka *et al.*, 2011) in order to build the intrinsically motivated system in this work. This system is comprised of an embodied actor which is connected to a predictor, that acts as the actor's memory and intrinsic motivation. The actor is implemented as a Restricted Boltzmann Machine which learns by using the Free-Energy reinforcement learning method (p. 19), or in its critical version, by the Adaptation to criticality method (p. 24). The predictor is implemented as an Echo-State Network and learns on-line using the Recursive-Least-Squares algorithm (p. 18). The predictor's input are the observation and the action received at each time step, and its output are the predicted next state and reward. At each time-step, it passes its activations, as well as a scalar reward signal, to the actor (see figure below, shown again for clarity). The reservoir activations, together with the environmental observation, act as a Markovian state for the actor. The internal reward signal together with the reward coming from the environment, are used for the learning of the actor. The rest of this sub-section describes the how the learning procedures of the ESN, the non-critical RBM and the critical RBM function together.



Schematic view of the actor-predictor architecture. The actor receives state and external reward from the environment, as well as an internal reward signal from the predictor. It uses the combination of the rewards for learning. It uses the predictor's state as memory in order to choose the next action to perform.

3.5.1 The Predictor

The predictor updates its states, as described in Section 3.2 (p. 1717)., by receiving training data from a RL setting. At each time-step $\boldsymbol{t}$, the ESN updates its history $\boldsymbol{h}$ with a training sample $\boldsymbol{h_t = (in_t, out_t)}$ received from the actor, where $\boldsymbol{in_t = [obs_{t-1}; a_{t-1}]}$ are the observation and action executed by the actor at time $\boldsymbol{t-1}$, and $\boldsymbol{out_t = [obs_t; r_{ext_{t-1}}]}$ are the observation and reward received by the actor from the environment at time $\boldsymbol{t}$.
The network then updates its reservoir state $\boldsymbol{m_{t+1}}$, driven by $\boldsymbol{in_t}$, and passes it to the actor. The reservoir state is defined as:

$$\boldsymbol{m_{t+1} = \sigma_g(W_{in} in_t + W m_t + v) + b} \qquad (\mathbf{3.5.1})$$

With $\boldsymbol{b = -xW}$, where $\boldsymbol{\sigma_g}$ is the sigmoid function with a gain $\boldsymbol{g}$, *v* is a noise term, and $\boldsymbol{b}$ is a memory bias with the 1-length vector $\boldsymbol{x = 0.5}$. The bias term offsets the mean value of the previous reservoir state being 0.5, ensuring that it would become 0, hitting the

center of the sigmoid function. The gain for the sigmoid is set to 4, in order to mimic the behaviour of a hyperbolic tangent function.

Once the ESN updates its reservoir, it calculates its learning improvement $\boldsymbol{l_t}$ as the difference between the Mean-Square-Error of the network's outputs for the last 1000 time steps before and after being trained on the last time step. It is defined as:

$$\boldsymbol{l_t} = \boldsymbol{P_q^{W_{t-1}\,out}} - \boldsymbol{P_q^{W_t\,out}} \tag{3.5.2}$$

with $\boldsymbol{P_q^{W_x\,out}} = \frac{1}{1000}\left(\sum_{i=\tau}^{t}\left(\hat{y}_i^{W_x\,out} - y_i\right)^2\right)$, τ = t-1000 and $\hat{\boldsymbol{y}}_i = \boldsymbol{f^{out}}(\boldsymbol{W_x^{out}}\boldsymbol{m_t})$,

where $\boldsymbol{P_q^{W_x\,out}}$ denotes the predictor's quality at time $\boldsymbol{x}$, $\boldsymbol{W_{t-1}^{out}}$ and $\boldsymbol{W_t^{out}}$ are the output weights of the ESN being trained on training samples $(\boldsymbol{in_{t-1}}, \boldsymbol{out_{t-1}})$ and $(\boldsymbol{in_t}; \boldsymbol{out_t})$ respectively, $\hat{\boldsymbol{y}}_i = [\widehat{\boldsymbol{obs}}_i; \hat{\boldsymbol{r}}_{\boldsymbol{ext_{i-1}}}]$ are the predicted observation and reward at time $\boldsymbol{i}$, $\boldsymbol{y_i}$ = $[\boldsymbol{obs_i}; \boldsymbol{r_{ext_{i-1}}}]$ are the received observation and reward at time $\boldsymbol{i}$, and $\boldsymbol{f^{out}}$ is the hyperbolic tangent function (see *Fig. 9* for high-level depiction of the calculation of the reward).

In order to update its output weights $\boldsymbol{W_t^{out}}$, the RLS algorithm[3] is ran by feeding it with $\boldsymbol{m_t}$ as input signal, and $\boldsymbol{y_t} = (\boldsymbol{f^{out}})^{-1}\boldsymbol{out_t}$ as output signal, with $(\boldsymbol{f^{out}})^{-1}$ the inverse hyperbolic tangent function.

Once $\boldsymbol{l_t}$ is calculated, an internal reward signal is passed to the actor as:

$$\boldsymbol{r_{int_t}} = \tanh \boldsymbol{l_t} \text{ if } \boldsymbol{l_t} \geq \boldsymbol{0} \text{ and } 0 \text{ otherwise} \tag{3.5.3}$$

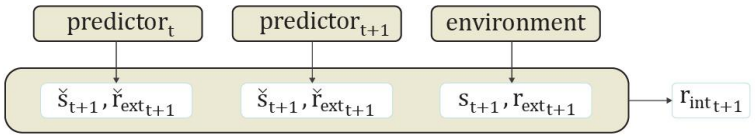


Fig. 9: High-level description of the generation of the intrinsic reward, if it is calculated one time step in history: it is the difference between the prediction error of the predictor before and after seeing the data.

The training of the ESN is done on-line, unlike in the originally proposed architecture[4] (Otsuka, 2010; Otsuka *et al.*, 2011) because the internal reward signal is implemented according to Schmidhuber's intrinsic motivation framework. The signal represents learning improvement, therefore the predictor needs to be learning simultaneously with the actor.

[3] *A time period of 1000 steps was chosen for the calculation of the MSE due to memory limits, which is in principle suboptimal according to the creativity framework's principle that pure intrinsic motivation should include all history of the learning system.*

[4] *No description of the training of the ESN was given in the paper with the proposed predictor-actor architecture which employed an ESN. The paper, which proposes the same architecture with the deployment of an RNN-predictor, uses off-line training, therefore the author assumes that the training of the ESN was done off-line as well.*

The ESN-predictor is poised to criticality by tuning its hyper-parameters according to the results from the Determination of Criticality using Fisher Information method (p. 2222 – Section 3.4.1).

3.5.2 The Actor

The actor has two learning procedures for its critical and non-critical version. The non-critical actor follows the Free-Energy RL training method (p. 19), whereas the critical actor folows the adaptation to criticality learning rule (p. 24). Both procedures incorporate the predictor's signals.

At each time-step $\boldsymbol{t}$, the non-critical actor receives an observation $\boldsymbol{obs_t}$ from the environment. It uses the memory from the predictor $\boldsymbol{m_t}$ to form the Markovian state $\boldsymbol{s_t} = [\boldsymbol{obs_t}; \boldsymbol{m_t}]$ [5]. The actor then chooses an action based on the softmax action selection (p. 19, eq. 3.3.8), and receives the next observation $\boldsymbol{obs_{t+1}}$ and the reward $\boldsymbol{r_{ext_t}}$ from the environment. $\boldsymbol{obs_t}$, $\boldsymbol{a_t}$, $\boldsymbol{obs_{t+1}}$ and $\boldsymbol{r_{ext_t}}$ are passed as the training sample $\boldsymbol{h_t}$ to the predictor.

The actor receives $\boldsymbol{m_{t+1}}$ from the predictor and uses $\boldsymbol{s_{t+1}} = [\boldsymbol{obs_{t+1}}; \boldsymbol{m_{t+1}}]$ to choose action $\boldsymbol{a_{t+1}}$. It then uses the reward $\boldsymbol{r_t} = \boldsymbol{r_{int_t}} + \boldsymbol{r_{ext_t}}$ in order to learn according to the FERL method. The memory $\boldsymbol{m_0^{e_i}}$ where $\boldsymbol{e_i}$ is a training episode and $\sum_{i=0}^{E-1} \boldsymbol{e_i}$ are all training episodes, is set to $\boldsymbol{m_{T-1}^{e_{i-1}}}$ for $\boldsymbol{i = 1, \ldots, E-1}$, following the creativity framework's principle to use as much of the learning system's history as possible (Schmidhuber, 2010).[6] See Algorithm 3 for the full learning procedure.

3.5.2.1 The Critical Actor

The critical actor's embodiment is accommodated in order to make use of the predictor's reward and memory. The network's sensors $\boldsymbol{S_i}$ take the values of the input $\boldsymbol{I_i}$ which is defined as

$$\boldsymbol{I_i} = [\boldsymbol{obs_{i+1}}; \boldsymbol{m_i}; \boldsymbol{r_i}] \tag{3.5.4}$$

where $\boldsymbol{obs_{i+1}} = [\boldsymbol{obs_{b_1}}, \ldots, \boldsymbol{obs_{b_7}}]$, with $\boldsymbol{obs_{b_i}}$ in {-1,1}, is the binarised observation from the environment received after the step of the network,
$\boldsymbol{m_i} = [\boldsymbol{m_{i_1}}, \ldots, \boldsymbol{m_{i_{Nr^P}}}]$, with $\boldsymbol{Nr^P}$ the number of reservoir units of the predictor, is the binarised memory $\boldsymbol{m_i}$ of the predictor,
and $\boldsymbol{r_i} = [\boldsymbol{r_{b_1}}, \boldsymbol{r_{b_2}}]$, with $\boldsymbol{r_{b_i}}$ in {-1,1}, is the binarised reward $\boldsymbol{r_t}$.

---

[5] *The rationale behind updating part of the visible units of the RBM with non-binary values, regardless of the fact that the RBM's units are binary in principle, followed from personal communication with Makoto Otsuka, the author of the paper with the original architecture. Binarising the predictor's activation units would generally result in loss of information and worse performance*

[6] *The original paper does not specify whether the predictor's states are being updated or not, because of no information about its training.*

Equivalently to the move during a time-step in the non-critical RBM's learning, the critical RBM performs a move as part of each sequantial Glauber update of its units. The move occurs at a random iteration $\boldsymbol{i}$ during the update, with $\boldsymbol{i} = (\boldsymbol{-1}, \ldots, \boldsymbol{N})$ and $\boldsymbol{N}$ is the number of units of the network. Similarly to the modified training of the non-critical RBM (Alg. 3)., the move in the training of the critical network handles the interaction between the predictor and the actor (see Algorithm 4). The memory is not reset at the start of each learning episode, but keeps getting updated.

Algorithm 3: Non-critical actor learning

Initialise weights and biases
Initialise the predictor
For each $\boldsymbol{e}$ ($\boldsymbol{0} \leq \boldsymbol{e} < \boldsymbol{E}$) do
    $\boldsymbol{t} \leftarrow \boldsymbol{0}$
    Reset environment and collect initial $\boldsymbol{obs\_t}$
    $\boldsymbol{if\ e = 0}$ do
        Initialise predictor's memory $\boldsymbol{m_t}$
    else
        $\boldsymbol{m_t} \leftarrow \boldsymbol{m_T}$
    Choose action $\boldsymbol{a_t}$
    Calculate $\boldsymbol{Q_t}$
    Repeat
        Collect $\boldsymbol{m_{t+1}}$ from the predictor
        Collect observation $\boldsymbol{obs_{t+1}}$ and reward $\boldsymbol{r_{ext_t}}$.
        Update the history $\boldsymbol{h_t}$ of the predictor
        Collect $\boldsymbol{r_{int_t}}$ from the predictor
        Choose action $\boldsymbol{a_{t+1}}$
        Calculate $\boldsymbol{Q_{t+1}}$
        Calculate temporal weights and biases updates using (*eq. *)
        $\boldsymbol{t} \leftarrow \boldsymbol{t+1}$
    until the goal is reached or $\boldsymbol{t = T}$
    Update the weights and the biases using (*eq.*)
end for

Algorithm 4: Move in a critical actor's learning

If $\boldsymbol{i}$ encodes step do
    Choose action $\boldsymbol{a_t}$
    Collect $\boldsymbol{m_t}$ from the predictor
    Collect observation $\boldsymbol{obs_{t+1}}$ and reward $\boldsymbol{r_{ext_t}}$
    Update the history $\boldsymbol{h_t}$ of the predictor
    Collect $\boldsymbol{r_{int_t}}$ from the predictor and calculate $\boldsymbol{r_t}$
    $\boldsymbol{S_i} \leftarrow [\boldsymbol{obs_{t+1}}; \boldsymbol{m_t}; \boldsymbol{r_t}]$
    Set $\boldsymbol{t} \leftarrow \boldsymbol{t+1}$

4 Implementation

This project was implemented in Python, with the exception of a Matlab script used for the Determination of the Edge of Criticality algorithm for the echo-state network.

The research work demanded the complete implementations of two of the methods used in the architecture: the free-energy RL method for the learning of the non-critical RBM (Sallans and Hinton, 2004) and the algorithm for Determination of the Edge of Criticality of the ESN (Livi, Bianchi and Alippi, 2018). The first method was modified in order to account for the presence of an additional reward signal and use it in the learning.

An implementation of the augmented training algorithm for the ESN (Jaeger, 2003) and a reproduction of the observed results were performed during the building of the predictor. The implementation of the FERL approach involved the connection of the RBM to the Frozen Lake environment in OpenAI Gym (OpenAI, 2016).

An existing ESN package (Korndörfer, 2015) was modified in order to perform on-line learning with RLS (Lewis, 2020a), as well as to incorporate it within the predictor-actor architecture.

The theoretical creativity framework for maximising intrinsic reward (Schmidhuber, 2010) was implemented into all elements of the architecture: the ESN involved the generation of an intrinsic reward and both learning methods of the RBMs incorporated the reward signal according to the creativity algorithm.

The implementation of the Adaptation to Criticality method in Python was freely available (Aguilera, 2018). It was modified in accordance with the described above: to connect to a new environment, as well as to interact with a predictor.

Finally, the research work involved the implementation of the environment used for the experiments.

The following section gives details about the implementations.

4.1 The ESN

A readily available implementation of an echo-state network was used and adopted for the purposed of the project (Korndörfer, 2015). In order to test the modified implementation of the echo-state network, the author reproduced the results on the 10-th order benchmark task NARMA (Jaeger, 2003):

$$\boldsymbol{d(n+1) = 0.3d(n) \ + \ 0.05d(n)\sum_{i=0}^{9} d(n-i) + \ 1.5u(n-9)u(n) + 0.1 \ (4.1.1)}$$

where the input to the system $\boldsymbol{u}$ is an i.i.d. input sequence sampled from an uniform distribution, and the task is to predict the next output.

NARMA is a nonlinear auto-regressive moving average task which is a benchmark for testing memory in system identification with RNNs because it has a long time-lag. It was chosen as benchmark for the ESN in this project because the Letters environment that the ESN is trained on in this project's experiments is a similar long-term dependency problem.

4.1.1 Training

The only available work with results for the performance of an ESN on NARMA involves the use of an augmented training algorithm. It consists in the calculation of a squared version of the network state, defined as the $\boldsymbol{2N \ + \ 2 \ \times \ 1}$ vector:

$$\boldsymbol{x_{squares}(n) = \left[u(n), x_1(n), \dots, x_N(n), u^2(n), x_1^2(n), \dots, x_N^2(n)\right]} \qquad (4.1.2)$$

The update equation for the network state remains the same (p. 17 - eq. 3.2.2), but the computation of the outputs uses its squared version instead:

$$\boldsymbol{y(n+1) = f^{out}\left(w_{squares}^{out} x_{squares}(n+1)\right)} \qquad (4.1.3)$$

where $\boldsymbol{w_{squares}^{out}}$ is the length $\boldsymbol{2N \ + \ 2}$ output weight vector.

The rest of the offline training procedure *(p. 18),* stays unchanged.

4.1.2 Results

The author implemented the 10-order system and the augmented training algorithm from scratch, and followed the training procedure and the hyper-parameters used in the paper to train the ESN. The network's reservoir was set with spectral radius 0.8 and its sparsity was 0.95. An i.i.d. input sequence sampled from the uniform distribution over [0, 0.5] was used to drive the network. It was run for 1200 time steps with uniform noise of size 0.0001. The transient was set to 200 time steps. The testing sequence was of 2200 steps with the same transient. The test error used was a Normalised-Mean-Squares Error.

Table 1 presents the results showed in the paper and the results obtained by the author. The mean results were obtained by calculating the mean NMSE of 100 instantiated

networks driven by a generated input-output training sequence of the specified. The results are close to the reference ones. The difference might be due to initial conditions of the generated NARMA signal. Running the networks with a newly generated input led to a slightly bigger error. There was some variability in the mean NMSE of multiple runs with 100 networks, so averaging the error over more networks would potentially lead to values closer to the reference ones. The differences for the networks of size 100 and 400 are both ~0.01 which shows consistency. The bigger difference for the 50-unit network might be due to higher sensitivity to initial conditions of the driving system due to the smaller training size.

Parameters	*Reference NMSE*	*Reproduced NMSE*
50 units, 1000 points	0.084	0.1369
100 units, 1200 points	0.032	0.0421
400 units, 4000 points	0.0098	0.0219

Table 1: Reproduced results for NARMA using the augmented reservoir algorithm, before and after running the FIM method for criticality

4.2 FIM Maximisation

The source code for the Determination of the Edge of Criticality using FIM maximisation algorithm was not available, therefore the procedure was implemented by the author from scratch, which involved a few deviations from the paper with the proposed method.

Following the definitions of the matrices in the equation for the estimation of the FIM from Section III. of the paper (Livi, Bianchi and Alippi, 2018) resulted in some inconsistencies in their dimensions, so the author consulted the reference paper where the methods were stated to be taken from (Berisha and Hero, 2015). After consolidating the two, the definition of the matrix R (p. 22 – eq. 3.4.3) from the reference paper was taken.

The authors' new formulation for the constraints of the convex optimisation problem which is stated in the Appendix (Livi, Bianchi and Alippi, 2018) resulted to be of little use for the successful solution to the original semidefinite problem (p. 2222, eq. 3.4.4). What was used in the project was the original formulation which the authors referenced (Berisha and Hero, 2015). The optimisation problem was implemented using CVX – a Matlab-based modeling system for convex optimisation, which allows for straight-forward implementation of advanced convex programming (CVX Research, 2020). The specific model implementing the optimisation problem used indexed dual variables which allowed for the assignment of separate constraints on the elements of the FIM matrix in a for-loop.[7] The Matlab Engine API for Python (MATLAB & Simulink, 2020) was used in order to run functions using Matlab's CVX package within a Python session. The implementation required correct mappings between types of variables in the two languages.

[7] Since this project is written in Python, Python's package for convex optimisation CVXOPT was considered first to be used for the implementation of the problem. However, the functions from Python's software demanded a complex mathematical reformulation of the problem. Consequently, the author decided to use Matlab's package instead.

4.2.1 Training

The task NARMA was used again, in order to test whether the critical configuration of the parameters of the network resulting from the algorithm for determining the edge of criticality would improve the performance reported for the non-critical network.

The ESN was tuned with the best parameter configuration found by the algorithm to determine the edge of criticality. The algorithm was used to optimise 3 hyper-parameters: the spectral radius $\boldsymbol{\rho(W)}$ and the sparsity $\boldsymbol{s}$ of the reservoir, and the input weights scaling $\boldsymbol{a}$. The number of units in the reservoir was set to 100. The parameter configuration values for the spectral radius were 5 values in the interval of [0,4; 1] because the spectral radius is sufficient to be below unity in order to preserve its echo-state property (Jaeger, 2003). The sparsity and the input scaling were 10 values each in the intervals of [0.3; 0.9] and [0,3; 0,8] respectively. This resulted in a total number of 500 configurations to be evaluated. The number of independent trials for each parameter configuration was T = 5 and the number of perturbations made for each trial was M=20. These values are smaller than the ones used in the original experiments, due to the limited time and computational power which were available to run the algorithm. The input signal was generated from an uniform distribution over [0; 0.5] with coefficient 1.5 before the third term (see p. 3131, eq. 4.1.1) .The network states were the squared version of the states (see p.31, eq. 4.1.2). The best parameter configuration found by the algorithm for this run $\boldsymbol{r_0}$ is shown in the first row under run in Table 2, together with the performance error.

	N_x	$\rho(W)$	s	a	NMSE
r_0	50	0.93	0.43	0.3	0.0547
r_1	50	0.67	0.63	0.3	0.1344
r_2	50	1.47	0.43	0.52	2.4554
r_3	50	1.47	0.77	0.58	3.3757
r_4	50	0.67	0.37	0.47	0.1069

Table 2: Runs with the FIM method showing the chosen parameter set and the NMSE error

A number of additional runs were done with the criticality algorithm to explore different possibilities to find an optimum parameter set. In these runs the same configurations for $\boldsymbol{r_0}$ were used with the exception of:

$\boldsymbol{r_1}$ - The input signal was generated from an uniform distribution over [0; 1], like presented in (Livi, Bianchi and Alippi, 2018). In order to preserve the stability of the NARMA output signal and keep it bounded between 0 and 1, the coefficient before the third term (see p. 3131, eq. 4.1.1) was set to 0.4.

$\boldsymbol{r_2}$ - The number of trials was set to T = 10 and the perturbations = 80, like in the training in the paper, but for less parameter configurations:

$\boldsymbol{r_3}$ - 5 values for the spectral radius were taken from the interval [1; 1.6] based on data that the spectral radius of the ESN does not need to be under unity in order to guarantee its echo-state property (Jaeger, 2003).

$\boldsymbol{r_4}$ - A training input sequence of length 2500 was used instead of the length of 1200 used in run 0. The training length used by the authors (Livi, Bianchi and Alippi, 2018) was not specified.

4.2.2 Results

As can be seen from the results, the critical configurations found by the runs did not achieve better results than the ones reported in Table 1. This is not an unexpected result considering the fact the FIM determinant maximisation method for the task NARMA has a reported correlation value of only 0.52 with the parameters found by maximising the performance (Fig. 10).

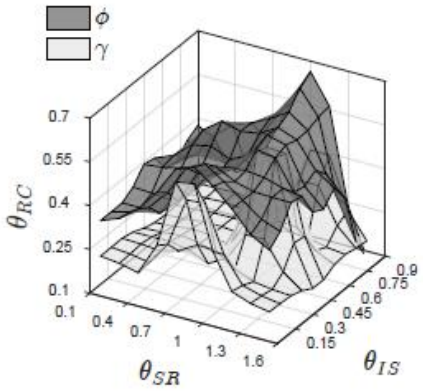


Fig. 10: Graphical representation for the computed edge of criticality. The light grey manifold represents the parameter configurations that maximise prediction accuracy and the dark grey manifold are the configurations where the FIM is maximised (Livi, Bianchi and Alippi, 2018)

Although this value is higher than the other two approaches to find the edge of criticality, it is still close to random chance. It might be of importance that the FIM algorithm is ran on only one input-output training sequence and generalising on more runs would improve the performance. However, improving the proposed algorithm is not in the scope of this project. Another reason for lower performance would be sensitivity to network configuration. Although the authors guarantee robustness regarding this issue, if the scope of this work allowed, it would be useful to test.

4.3 The Free-Energy Principle

The FE approach was implemented from scratch. In order to test the implementation, the network was ran with the toy environment from OpenAI Gym Frozen Lake (OpenAI, 2016). The paper where the method was originally proposed (Sallans and Hinton, 2004), shows that the FERL approach achieves close to classic Q-learning performance for small, and better performance for bigger, tasks with factored states and actions. The implemented RBM in this work did not achieve the performance of a Q-learning algorithm. The received rewards were sparsely spread during the episodic runs, and did not show a trend of learning. The Q-values of the environment states seemed to grow proportionally, without the values of the better states becoming significantly bigger than the values of the other states. Optimisation of the learning rate and the inverse temperature did not result in better performance. The actor was ran together with a predictor, as described in Section IV. Joint Architecture, but it did not improve the performance. A reason for this might be the fact that Frozen Lake is a task too simple for the FERL method. Its states and actions are not multi-factored, therefore a Q-table to store the actual Q-values is feasible, and estimation of these values leads to worse results.

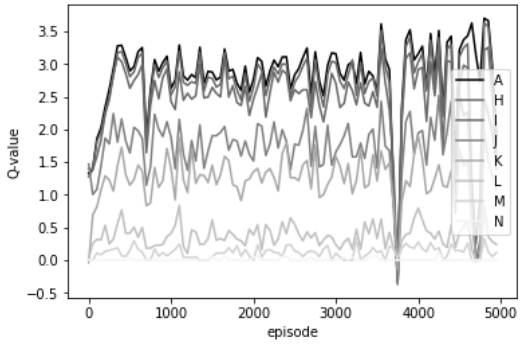


Fig. 11: Q-value of each state averaged over 50 episodes. State A has cost 1 and state N has cost 10. States of lower cost have higher Q-values than the states of higher cost.

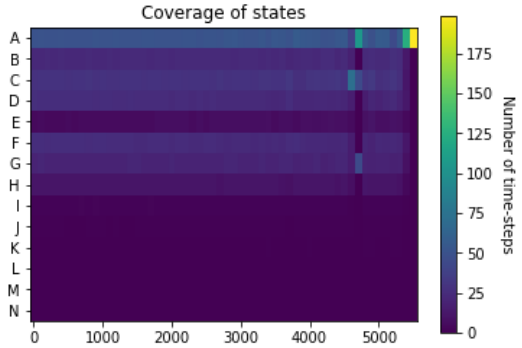


Fig. 12: Number of times the states were reached, averaged over 100 episodes. The states of lower cost were reached the most frequently

4.3.1 Results

On the Letters environment (see p.39, Section 5.1*),* the Q-values for the separate states showed similar behaviour. The Q-values for the more frequently visited states grew higher than these for the less frequently visited states (Fig. 11). It seemed that the coverage of environmental states was too imbalanced in order for the Q-values to converge (Fig. 12). Due to the temporal dependency of the task, it is natural that the RBM did not achieve the goal state frequently without a memory entity. With only the last state visited as input, it does not have the necessary information to consider the previously traversed states, part of the goal state combination, to be as beneficial for maximising the reward.

After ~5400 episodes, the actor starts performing the same action, leading to reaching state A only (Fig. 12). This leads to the steep growth of its Q-value, leading to numerical instability in the action selection of the network.

In order to test the method when the network is presented more frequently with the goal state, the actor was fed with the goal state action sequence multiple times for multiple episodes. In this case, the Q-value of the goal state grew significantly more than for the other states, while the average Q-value per episode kept growing, like the behaviour of the network on the toy environment (Fig. 13). The weights and the biases grew very significantly, up to values of 8 and 11, which led to loss of stability in the network. Due to the big weight values, the probability of the estimated hidden unit values became very close to (or) 1, leading to the entropy being close to 0 (or undefined). With entropy close to 0 and very big log-likelihood of the states, the Q-values became very high, which led to instability in the logistic sigmoid function in the action selection process (see Fig. 14 and 15).

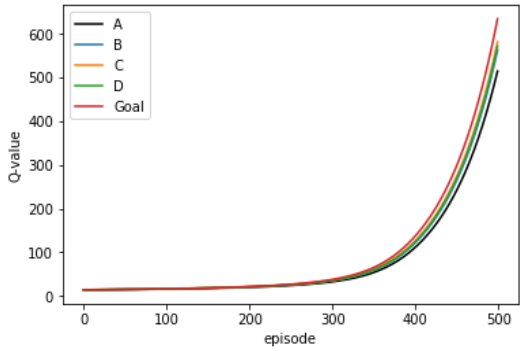


Fig. 13: Q-values of the first 4 states, and the goal state for run with the goal action sequence for 500 episodes. The Q-values of the goal state is the highest, but all Q-values grow steadily.

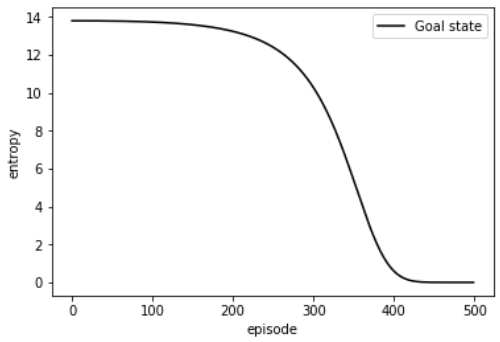


Fig. 14: The entropy of the goal state diminishes, reaching 0.

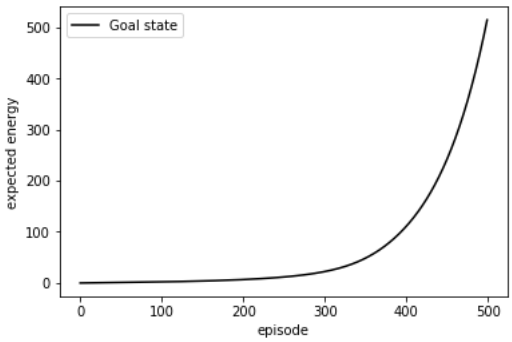


Fig. 15: The expected energy of the goal state rises significantly.

In the context of the Frozen Lake toy environment, the negative growth of the free-energy (and the positive growth of the Q-values respectively), which was leading to the loss of stability in the action selection, was prevented by setting an upper boundary on it. This, however, did not lead to improvement in performance. With the Letters environment, the use of L-2 regularisation was implemented instead, which led to slower growth of the weights and the biases. They still led to instability in the action selection, but on a later stage in the learning process.

4.3.2 Discussion

It seems that the Q-values grow steadily with each episode, due to the overall growth of the log-likelihood of the state probabilities and the diminishment of the entropy due to more confidence in estimating the values of the hidden units. The receipt of more rewards accelerates this growth. This follows from the specifics of the free-energy training method. A reason for the unsatisfactory performance of the implementation might be the hyper-parameters such as learning rate and inverse temperature, or the use of different regularisation methods or training techniques. Evaluation and potential improvement of the training presented by the authors of the method was, however, out of the scope of this project. The results presented for the non-critical RBM are therefore biased by these challenges.

The behaviour of the RBM was similar for the Frozen Lake toy environment and the Letters environment. This might look odd considering the different difficulties of the tasks, however, the reason for the behaviour might be the fact that during both tasks, the network was only rarely presented with the reward.

4.4 Adaptation to Criticality

The criticality method's source code for the RBM was publicly available and was written in Python (Aguilera, 2018), therefore the method did not need to be implemented by the author. The functions with the learning rules remained unchanged, however, the code was modified in order to incorporate a predictor network and to therefore introduce memory and intrinsic reward into the adapting system.

The changes related to the independent actor involve the removal of the weights linking units within each layer of the network, in order to make it restricted. This was done with the goal to make the comparison between the two training methods of the actor consistent, since the non-critical method makes use of a restricted Boltzmann Machine. The critical method is robust to the specific architecture of the Boltzmann Machine (Aguilera and Bedia, 2018), therefore this change doesn't affect the training procedure. In order to confirm the successful adaptation to criticality of the new architecture, it was trained on the Open AI Gym Frozen Lake toy environment, after which the source code for showing the signs of criticality was run, indicating the same transitions in the entropy of the system as in the original experiments.

5 Experiments

In order to test for the influence of criticality on the behaviour of the proposed intrinsically motivated architecture, the project involves the design and implementation of a simple environment whose state-action space allows for adaptive behaviour. The research hypotheses are tested on the task by running a set of experiments with a 2 x 2 design, optimising the parameters for each model separately. The following describes the task and the training of the models.

5.1 The Letters Task

The experiment in this work is a toy scenario where the agent learns different sequences of actions. Since the states in the environment are comprised of action sequences of varying length, their difficulty to achieve naturally grow proportionally to their actions length. The idea behind the hierarchically generated action is that a system traversing the states will start learning how to achieve simple states of shorter sequences of actions and will gradually improve on what it's learned by reaching more complex states. The task is deterministic but it allows for multiple ways of reaching states. It is also time-dependent since the output of each time step depends on the actions generated in the previous time steps. The scenario imposes the need for a good exploratory mechanism in order to cover more states, as well as good memory in order to link states and actions to their outcomes at different moments in time.

Precisely, there are 13 states of costs in the range [1; 10] present in the environment. The cost represents the number of actions that need to be taken in order to reach the state. The actions that can be taken to move from state to state are four. They also represent states – performing action A lends the system at state A. These four states are available to the system at any time step. The higher the cost of a state, the more time steps need to be executed to reach it. Each state is a combination or a number of combinations of a varying number of states of smaller costs. Two states or two combinations can't have the same action sequence. A state of a particular combination means it can be reached via all the combinations which its sub-states have. The total number of action sequences present in the environment is 93. The goal state is state N of cost 10. It has a total number of 25 action sequences. The environment returns external reward 1 if the goal state is reached and 0 otherwise. Table 5 shows the first 20 states of the environment. See the Appendix for the full list of states (p. 5353). The states highlighted in yellow are also the actions.

The environment's step function returns as observation at time step $\boldsymbol{t_i}$ the state of the highest cost which was reached. For example, if the system performs actions $\boldsymbol{a_1}$= B, $\boldsymbol{a_2}$ = A, $\boldsymbol{a_3}$ = D it will receive observations $\boldsymbol{o_1}$ = A, $\boldsymbol{o_2}$ = B and $\boldsymbol{o_3}$ = H. Note that $\boldsymbol{a_2}$ followed by $\boldsymbol{a_3}$is the state G of cost 2, but since at $\boldsymbol{t_3}$ H is achieved as well, which is of higher cost – $\boldsymbol{c_H}$ = 3, H is returned as observation. The underlying state matrix would have at time step $\boldsymbol{t_3}$ states D, G and H since they were all available at $\boldsymbol{t_3}$.

The environment notifies the system when the goal state was reached. The initial observation for $\boldsymbol{t_0}$ is -1. The environment is reset at the end of each episode by resetting the tracking of performed actions.

State	*Cost*	*Combination*	*State sequence*
A	1	A	A
B	1	B	B
C	1	C	C
D	1	D	D
E	2	BB	BB
F	2	DC	DC
F	2	DD	DD
F	2	AB	AB
G	2	AD	AD
G	2	AC	AC
G	2	CB	CB
H	3	BG	BAD
H	3	BG	BAC
H	3	BG	BCB
H	3	DAC	DAC
I	4	HD	BADD
I	4	HD	BACD
I	4	HD	BCBD
I	4	HD	DACD
I	4	CAAB	CAAB

Table 3: The Letters Task with states, combinations and unique action sequences

5.2 Training

The experiments in this work follow the 2 x 2 design method (table 3), where the models are the following:

1) An RBM-actor adapting to criticality, connected to a poised at criticality ESN-predictor
2) An RBM-actor adapting to criticality, connected to a non-critical ESN-predictor
3) An RBM-actor learning with free-energy, connected to a poised at criticality ESN-predictor
4) An RBM-actor learning with free-energy, connected to a non-critical ESN-predictor

Model	*RBM*	*Training*	*ESN*	*Training*
$\boldsymbol{A^*P^*}$	critical	AC	critical	RLS
$\boldsymbol{A^*P}$	critical	AC	non-critical	RLS
$\boldsymbol{AP^*}$	non-critical	FE	critical	RLS
$\boldsymbol{AP}$	non-critical	FE	non-critical	RLS

Table 4: The four models used in the experiments

The hyper-parameters of the non-critical ESN are chosen by training it with samples collected from the Letters environment with a random policy. The training is done with the online RLS algorithm, since this algorithm is used in the Letters task. The samples are comprised of two datasets – one with linear and one with binarised inputs and outputs. The number of reservoir units $\boldsymbol{N_x}$ used in the grid-search is between the values [50, 100, 150,200]. The spectral radius of the reservoir $\boldsymbol{\rho(W)}$ is in the range [0.4; 1], the sparsity of

the reservoir $\boldsymbol{s}$ is in the range [0.5; 0.9] and the input weights scaling $\boldsymbol{a}$ is in the range [0.5;1]. The readout layer $\boldsymbol{f^{out}}$ is optimised for an identity and a hyperbolic tangent function.

The number of reservoir units $\boldsymbol{N_x^*}$, the encoding of input data $\boldsymbol{in}^*$ and the readout activation function $\boldsymbol{f^{out}}^*$,optimised for the non-critical ESN, are kept the same for the critical ESN. Its spectral radius $\boldsymbol{\rho}^*(\boldsymbol{W})$, sparsity $\boldsymbol{s}^*$and input weights scaling $\boldsymbol{a}^*$ are selected by running the FIM algorithm. [8]

The optimised parameters for the non-critical RBM's training are the number of hidden units $\boldsymbol{N^h}$ (in the range [10;50]) and the inverse temperature $\boldsymbol{\beta}$ (either the constant value of 1 or increasing incrementally within [1;10]). [9]

For the critical RBM's training procedure, the used number of units $\boldsymbol{N^h}^*$ are the same as in the non-critical case. The inverse temperature $\boldsymbol{\beta}^*$ and the number of iterations per run $\boldsymbol{T}$ are these used in the original paper's training procedure.

The parameters for the training of each model are present in Table 4.

Parameters	*Values*	*Parameters*	*Values*	*Parameters*	*Values*
$\boldsymbol{in}, \boldsymbol{in}^*$	linear	$\boldsymbol{f^{out}}, \boldsymbol{f^{out}}^*$	identity	$\boldsymbol{\gamma}$	0.01
$\boldsymbol{N_x}, \boldsymbol{N_x^*}$	50	$\boldsymbol{\rho}^*(\boldsymbol{W})$	0.4	$\boldsymbol{\lambda}, \boldsymbol{\lambda}^*$	0.5
$\boldsymbol{\rho}(\boldsymbol{W})$	0.7	$\boldsymbol{s}^*$	0.37	$\boldsymbol{\beta}$	[0;15]
$\boldsymbol{s}$	0.9	$\boldsymbol{a}^*$	0.36	$\boldsymbol{\beta}^*$	1
$\boldsymbol{a}$	1	$\boldsymbol{N^h}, \boldsymbol{N^h}^*$	50	$\boldsymbol{T}$	1000

Table 5: Parameters for the conducted experiments

[8] *Using different than these three parameters would require making changes in the algorithm which is out of the scope of this work.*

[9] *An inconsistency was found in the use of the parameter inverse temperature between the two papers with the predictor-actor RL architecture* (Otsuka, 2010; Otsuka *et al.*, 2011)*. In the ESN-version of the architecture, the inverse temperature was used in the action selection of the RBM (*eq.*), but not in the calculation of the free energy (*eq.*), whether in the RNN-version, it was used in both. This inconsistency was not given an account for in the communication with the author of the paper. Since it is out of the current scope to tune the free-energy hyper-parameters or to investigate their usage, the ESN-version of the method was used in this work.*

6 Results

In order to assess the way the systems explore their environmental space under and without criticality, different aspects of the learning behaviour of the models were measured. They involve the behaviour of the actor in relation to coverage of states and exploration during hierarchical traversing of states. The behavior of the predictor is measured as its prediction improvement and in terms of its generated internal reward. The extent to which the goal state is reached is also investigated.

6.1 Coverage of States

In order to assess how fully the actors explore the state space, the coverage $\boldsymbol{C^{st}}$ was calculated as:

$$\boldsymbol{C^{st}} = \frac{\mathbf{1}}{\boldsymbol{E}} \sum_{\boldsymbol{e=1}}^{\boldsymbol{E}} \boldsymbol{x_e}\text{ , for i = 1, … ,E,} \qquad (6.1.1)$$

where $\boldsymbol{x_e}$ is the number of different states covered for $\boldsymbol{T}$=200 time steps. Number of steps are used in order to account for different-length episodes for the learning of the $\boldsymbol{A}$ and $\boldsymbol{A}^*$ models.[10]

Figures 16 and 17 show the results for the non-critical actor $\boldsymbol{A}$ and the critical $\boldsymbol{A}^*$.[11]

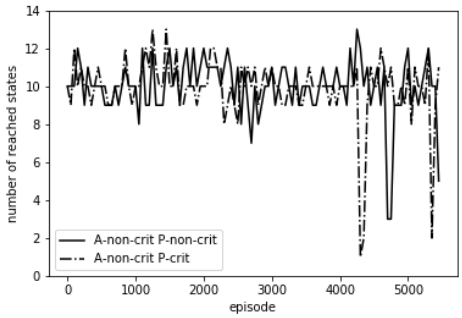


Fig. 16: Average covered states for 50 episodes for the two models of the non-critical actor.

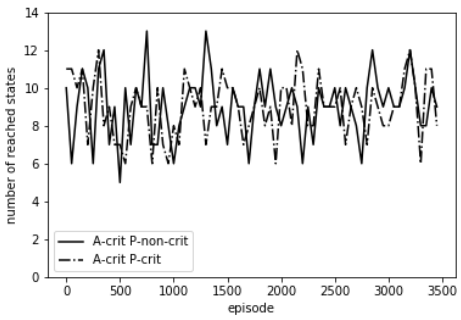


Fig. 17: Average covered states for 50 episodes for the two models of the critical actor.

Both the critical and the non-critical actors cover around 9 states per episode, with the critical model fluctuating more. The critical predictor does not seem to change these results. The first set of models achieve higher number of reached states with $\boldsymbol{P}^*$, while in the second set of models, the peaks belong to $\boldsymbol{P}$.

[10] An episode for the critical RBM involves 10 000 time steps, whereas an episode for the non-critical is 200 steps. The bigger value of time steps for A^* is important for the switch to a critical regime of behaviour.

[11] Due to the long duration of the run of the critical RBM training procedure and the limited time available for the experimental runs, the network was run for a smaller number of episodes than the non-critical one. Thus, the figures are shown separately for the two learning procedures. The non-critical RBM starts using stability after 5000 episodes, therefore the results are affected by this issue (*See Discussion in Actor Implementation*).

The drops in the last thousand episodes of the training of $\boldsymbol{A}$ are caused by the execution of the same action A for all time-steps, landing the network at state A.

There is no apparent influence of the critical predictor on the coverage of states. The critical actor varies more at this performance, but the number of states does not go up due to criticality. The coverage does not improve or worsen with the advance of the episodes.

6.2 Exploration during Hierarchical Learning

States of higher costs are harder to reach than states of lower costs, because they involve more time-steps and cover more actions, and exploration can create a trade-off between how many states of different costs are reached and how many different combinations of these states are reached. This trade-off might be shown by looking at the number of ways the system explores sub-states of states, where a sub-state here is a state which forms part of another state's combination.

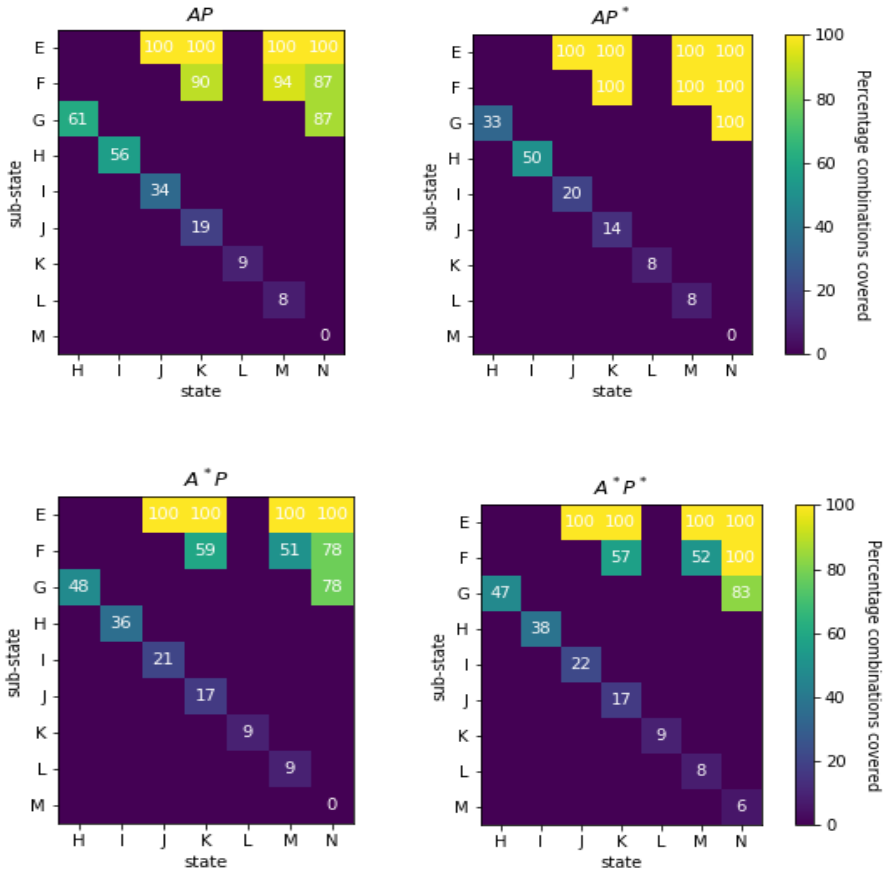


Fig. 18: Heat-maps showing pairs of sub-states and states. Each value on the y-axis is, if numbered, part of a combination of a state, on the x-axis. The values indicate the ratio between the number of ways the sub-states were reached and the total number of ways, for an episode, before reaching the state (which includes them in its combinations).

Fig. 18 shows the ratio between performed and total action sequences of each sub-state of a state. Each state on the x-axis is a complex state which has at least one of the states on the y-axis in its combinations. In the case of state-combination pair H-G, this map shows how much of the space of action sequences of state G was explored before reaching H for the first time in an episode. The pairs without a value indicate that they are non-existent, e.g. state I does not have state G in any of its combinations.

The $\boldsymbol{A}^*$ models perform slightly worse than the $\boldsymbol{A}$ models overall. They perform better on the sub-states of higher cost than the $\boldsymbol{AP}^*$. The $\boldsymbol{AP}$ model performs better than the other three on these. The $\boldsymbol{AP}^*$ is better than the rest on the sub-states of lower cost and in fact covers all three combinations of both states F and G before reaching M and N respectively. There is no difference between the two $\boldsymbol{A}^*$ models.

Sub-state E is always fully covered because it has only one combination. Sub-state M of state N is never covered at all, apart from an occasion in $\boldsymbol{AP}^*$. In general, the sub-states of lower cost are better covered, but this might be biased by the fact that the higher the cost of a state, the more combinations it has, and the more difficult it becomes to cover them all. For comparison, state M has 16 combinations, while states F and G have four each.

6.3 Predictor Quality

Influence of criticality on the predictor's quality might include not only the way the critical actor traverses the state-action space, but the critical configuration of the hyper-parameters of the predictor leading to better learning, The predictor quality is calculated as the Mean-Square-Error:

$$\boldsymbol{C}_{\boldsymbol{t}}^{\boldsymbol{W}_{t-1}} = \sum_{\boldsymbol{\tau=t-1000}}^{\boldsymbol{t}} (\hat{\boldsymbol{x}}_{\boldsymbol{t}} - \boldsymbol{x}_{\boldsymbol{t}})^{\boldsymbol{2}} + (\hat{\boldsymbol{r}}_{\boldsymbol{t}} - \boldsymbol{r}_{\boldsymbol{t}})^{\boldsymbol{2}}\,, \tag{6.3.1}$$

where $\boldsymbol{x}_{\boldsymbol{t}}$ is the observation received at time $\boldsymbol{t}$, $\boldsymbol{r}_{\boldsymbol{t}}$ is the reward and $\boldsymbol{W}_{\boldsymbol{t-1}}$are the weights updated at the previous time-step.

Figures 19 and 20 show the MSE for the $\boldsymbol{A}^*$ models and for the $\boldsymbol{A}$ models, averaged over every 200 episodes. They show no improvement of the predictor's quality. Both $\boldsymbol{P}^*$ are achieving worse performance than $\boldsymbol{P}$, suggesting that the critical tuning of the predictor did not lead to better performance. The actor is also not contributing to its improvement, contrary to the hypothesis that a particular state-action generation could have had influence on it.

The sharp drops in MSE in the second plot are due to these episodes where the actor kept reaching the same state during all time-steps, leading to the generation of very predictable training data.

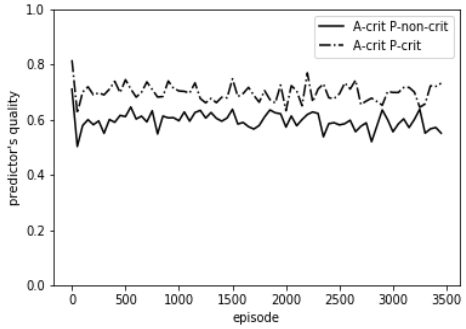


Fig. 19: MSE of the predictor during learning for the critical actors

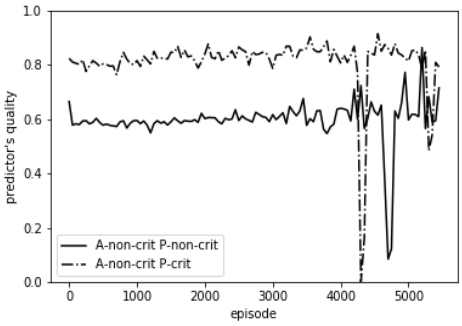


Fig. 20: MSE of the predictor during learning for the non-critical actors

6.4 Internal Reward

An intrinsically motivated system should maximise its internal reward. Figures 21 and 22 show the evolution of the intrinsic reward, averaged over every 200 episodes. The network does not show generation of more rewards. The values look too small for the successful learning of the system. The non-critical predictors exhibit slightly higher rewards but this difference is insignificant considering the small range. The drops in the reward between episodes 4000 and 500 are the episodes where there was coverage only of a single state, leading to the MSE of the predictor approaching 0.

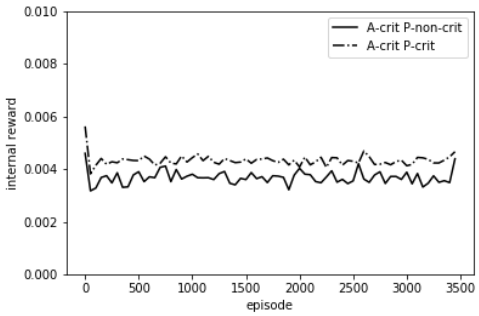


Fig. 21: Intrinsic reward which the actor receives in the critical actor models

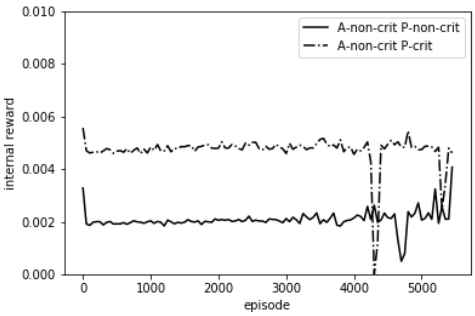


Fig. 22: Intrinsic reward of the non-critical actor models

6.5 External Reward

The experiments show the behaviour of a system which is both intrinsically and extrinsically motivated. The external reward is not achieved often (fig. 23). The critical-actor models achieve higher ratio of reached goal to total number of steps, but they do not show to receive the external reward more often with the progress of training.

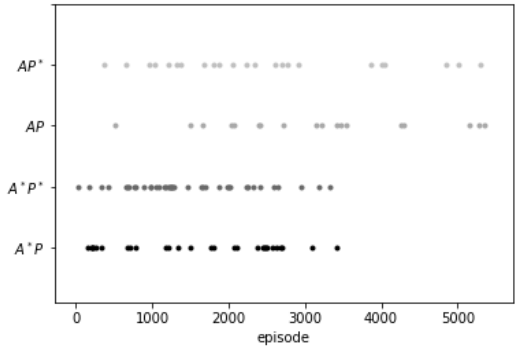


Fig. 23: The episodes when the goal state was reached for the four models. All are sparsely reached during the learning. The critical actors achieve better performance.

7 Discussion and Future Work

7.1 Rewards

The intrinsic reward is crucial for driving an intrinsically motivated system, but it nevertheless showed to be extremely small during the experimental runs. Alternative methods for bounding the learning progress in order to create the reward would result in better implementation of the intrinsic motivation. The present intrinsic reward seems too insignificant in order to drive interesting behaviour.

The external reward in the experimental runs was also sparse, due to the fact that the learners achieved the goal state very rarely. Therefore, it is not surprising that the system did not show significant learning in the results. A better implementation of the generation of the intrinsic reward, as well as more/bigger external rewards received from the environment would address this issue. An alternative design could keep the reward of the goal state to be 1 and of the base states to be 0, while setting all other with a reward of 0.5. This would encourage the actor to reach more complex states overall, not only to aim at the goal state.

7.2 Environment

Training the non-critical actor which estimates Q-values showed that the Q-values grew bigger for the base states of cost 1, which are also actions in the environment. This is a drawback of the design which can be overcome by making the environment a POMPD. It would return no observation if no complex state is achieved. This could ensure that the focus of the non-critical actor is on complex states only, preventing the Q-values from growing for the frequently seen base states.

An alternative for the environmental observations addresses a different issue: the loss of information about complex states achieved together with other complex states. The environment returns the state of maximum cost reached at a time-step meaning that the network receives no knowledge about a reached sub-state which is part of its combination, if it is reached at the same time-step. This results in the predictor potentially not being able to associate the performed sequence of actions to this particular state. It would be interesting to see if returning all of the states reached at the same time-step would change the behaviour of the learners.

The environment could also be improved in terms of implementation efficiency. The current implementation resulted in long runs and was an issue considering the limited time available for the experimental results.

7.3 Actor

The non-critical actor's FERL method has been the most challenging element of the project, due to the fact that it generated multiple numerical instabilities, and needed to be extended with the use of training techniques, such as regularisation. The improvement of its implementation and a more thorough investigation of its dynamics would be a first

point of reference, if this work was to be extended and improved. The huge growth of the Q-values could be addressed by setting an upper boundary, instead of penalising weight and biases growth.

An investigation into the use of the memory can be made, and whether its binarisation always results in worse performance. Exploring in finer detail the inverse temperature parameter, and looking into its effects, when used in front of the entropy term in the free-energy calculation, would be very fruitful. The Q-values progress for the separate states could be tracked, including, as well as, separately from the memory, to understand how the convergence of the actual Q-values happens. Additionally, the hidden unit activations could be investigated, in order to look for encodings of the environmental data in the network.

While criticality in the actor results in coverage of goal states without maximising an external reward, it did not prove to generate intrinsic behaviour, as suggested in the project's hypothesis. This could be due to the small values of the intrinsic reward, making the bias values of the sensor units insignificant. The fact that the critical RBM receives information about its environment via its sensor biases, rather than its sensor units, is a crucial difference between the critical and the non-critical learning methods in this work. The critical RBM might need changes in its training procedure in order to address positively the research hypotheses.

7.4 Predictor

The predictor might learn better and generate higher rewards, if it is fed with the generated actions as inputs and the next time-step's returned states as outputs. Using action-state training samples would be conceptually closer to a time-lag task like NARMA, and is conceptually better for the purpose of the Letters environment: to allow for learning what sequences of actions create desired patterns in an environment, and learning different sequences of actions, which can achieve the same patterns. The current state received does not add meaningful information, since it is the generation of specific actions in time, which causes the returned states.

The results clearly showed that the quality of the predictor was not improving with time. This supports the idea described above that it might be due to the input-output signal being too "noisy". It might also be a result of the particular implementation of the RLS algorithm which was used. The only available library for multivariate RLS calculates not only the weights as parameters, but an intercept, as well as removes the mean from the input and the output. Due to the limited time available to implement the online learning of the predictor, implementing the RLS algorithm from scratch was not feasible. In future work, different algorithms and implementations could be investigated.

The training of the predictor could be done firstly offline using random environmental samples, prior to the learning of the actor. It would then continue to learn online with the state-action data generated by the actor. This would allow for better performance and potentially bigger intrinsic rewards passed to the actor, but it abuses the idea of memory representation, since it would keep history of events before the start of the history of the actor. An alternative would be to train the predictor together with the actor, but in an

asynchronous way. This would preserve the network's role of memory, while benefitting from more robust offline training.

Finally, poising the ESN at the edge the criticality by maximising the FIM does not guarantee optimal prediction accuracy. As extension of this work, alternative criticality approaches could be investigated.

7.5 Experiments

Experiments with the lack of presence of an extrinsic reward could be performed, in order to test the research hypotheses for a purely intrinsically motivated learning system. A dynamic experiment could be conducted with two phases of training: train the network to maximise an external reward and achieve a certain goal; then train it to maximise an intrinsic reward in a changed environment, which allows the achievement of this goal in a different way. In this manner, it would be tested whether the maximisation of the intrinsic reward helps the system to adapt and find the alternative way to achieve the same goal. It would also test the exploration-exploitation trade-off of the system.

8 Conclusion

This research work was a challenging project. All of the theory behind the elements of the model architecture were material, which the author was unfamiliar with, prior to commencing the work on the project. Some of the training methods involved implementation from the ground up: tuning the echo-state network to criticality and doing reinforcement learning with free-energy. They involved unforeseen practical challenges, such as inconsistences in the description of the methods or incomplete description of the training procedures. Most of the elements of the model architecture were based on lesser known research. The chosen creativity framework was theoretical and did not provide guidance on the practical implementation of the intrinsic reward. The connection of the elements involved complex design and implementation matters. The Letters Task presented the challenge to achieve the right level of complexity for the model and for the research question. The author addressed all of these matters with limited expertise and experience, as well as with limited time available to explore the different options.

The experiments with the joint architecture do not show significant results, and there is room for improvement both on the design and the implementation level of the research work. However, the author feels that the project's objectives were met in the sense that all of the elements of the model architecture were implemented and combined successfully. The joint architecture addressed nicely the research question and the conducted experiments provided room to test the research hypotheses. The joint model architecture was ran on a toy scenario, which was also able to allow for the evaluation of the proposed hypotheses. Finally, the research question in this project was addressed in a consistent way. Most importantly, the project was fruitful in the sense that it generated multiple different ways in which the efforts to explore the research hypotheses could go.

9 References

Aguilera, M. (2018) *Adaptation to Criticality. Source Code*, *GitHub*. Available at: https://github.com/MiguelAguilera/Adaptation-to-criticality-through-organizational-invariance (Accessed: 25 May 2020).

Aguilera, M. and Bedia, M. G. (2018) 'Adaptation to criticality through organizational invariance in embodied agents', *Scientific Reports*, 8(1), pp. 2–12. doi: 10.1038/s41598-018-25925-4.

BCS (2015) *BCS Code of Conduct*.

Berisha, V. and Hero, A. O. (2015) 'Empirical non-parametric estimation of the Fisher information', *IEEE Signal Processing Letters*, 22(7), pp. 988–992. doi: 10.1109/LSP.2014.2378514.

Chialvo, D. R. (2018) 'Life at the edge: Complexity and criticality in biological function', *Acta Physica Polonica B*, 49(12), pp. 1955–1979. doi: 10.5506/APhysPolB.49.1955.

CVX Research (2020) *CVX: Matlab Software for Disciplined Convex Programming | CVX Research, Inc.*, *CVX Research, Inc.* Available at: http://cvxr.com/cvx/ (Accessed: 25 May 2020).

Jaeger, H. (2003) 'Adaptive nonlinear system identification with Echo State networks', *Advances in Neural Information Processing Systems*.

Korndörfer, C. (2015) *Echo State Networks in Python*, *GitHub*. Available at: https://github.com/cknd/pyESN (Accessed: 25 May 2020).

Lewis, C. (2020a) *An implementation of recursive least squares in Python*, *GitHub*. Available at: https://github.com/cannontwo/rls (Accessed: 25 May 2020).

Lewis, C. (2020b) 'On Multivariate Recursive Least Squares and Extensions', pp. 1–10.

Livi, L., Bianchi, F. M. and Alippi, C. (2018) 'Determination of the Edge of Criticality in Echo State Networks Through Fisher Information Maximization', *IEEE Transactions on Neural Networks and Learning Systems*, 29(3), pp. 706–717. doi: 10.1109/TNNLS.2016.2644268.

Lukoševičius, M. (2012) 'A practical guide to applying echo state networks', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTU, pp. 659–686. doi: 10.1007/978-3-642-35289-8-36.

MATLAB & Simulink (2020) *MATLAB Engine API for Python*, *MathWorks United Kingdom*. Available at: https://uk.mathworks.com/help/matlab/matlab_external/get-started-with-matlab-engine-for-python.html (Accessed: 25 May 2020).

Mora, T. and Bialek, W. (2011) 'Are Biological Systems Poised at Criticality?', *Journal of Statistical Physics*, 144(2), pp. 268–302. doi: 10.1007/s10955-011-0229-4.

O'Connor, P. *et al.* (2013) 'Real-time classification and sensor fusion with a spiking deep belief network', *Frontiers in Neuroscience*, 7(7 OCT), pp. 1–13. doi: 10.3389/fnins.2013.00178.

OpenAI (2016) *Frozen Lake Environment in OpenAI Gym*, *OpenAI Gym*. Available at: https://gym.openai.com/envs/FrozenLake-v0/ (Accessed: 25 May 2020).

Otsuka, M. (2010) 'Goal-Oriented Representations of the External World : A Free-Energy-Based Approach'.

Otsuka, M. *et al.* (2011) 'Solving POMDPs using Restricted Boltzmann Machines with Echo State Solving POMDPs using Restricted Boltzmann Machines with Echo State Networks', (June).

Runco, M. A. and Jaeger, G. J. (2012) 'The Standard Definition of Creativity', *Creativity Research Journal*, 24(1), pp. 92–96. doi: 10.1080/10400419.2012.650092.

Sallans, B. and Hinton, G. E. (2004) 'Reinforcement learning with factored states and actions', *Journal of Machine Learning Research*, 5, pp. 1063–1088.

Schmidhuber, J. (2010) 'Formal theory of creativity, fun, and intrinsic motivation (1990-2010)', *IEEE Transactions on Autonomous Mental Development*, 2(3), pp. 230–247. doi: 10.1109/TAMD.2010.2056368.

Singh, S. *et al.* (2010) 'IEEE TRANSACTIONS ON AUTONOMOUS MENTAL DEVELOPMENT 1 Intrinsically Motivated Reinforcement Learning: An Evolutionary Perspective', pp. 1–14. Available at: https://web.eecs.umich.edu/~baveja/Papers/IMRLIEEETAMDFinal.pdf.

Sutton, R. S. and Barto, A. G. (2018) *Reinforcement Learning: An Introduction*. Second edi. Cambridge, MA: MIT Press.

Tkacik, G. *et al.* (2009) 'Spin glass models for a network of real neurons', (1), pp. 1–15. Available at: http://arxiv.org/abs/0912.5409.

10 Appendix

10.1 The Letters Task

State	Cost	Combination	Action sequence
A	1	A	A
B	1	B	B
C	1	C	C
D	1	D	D
E	2	BB	BB
F	2	DC	DC
F	2	DD	DD
F	2	AB	AB
G	2	AD	AD
G	2	AC	AC
G	2	CB	CB
H	3	BG	BAD
H	3	BG	BAC
H	3	BG	BCB
H	3	DAC	DAC
I	4	HD	BADD
I	4	HD	BACD
I	4	HD	BCBD
I	4	HD	DACD
I	4	CAAB	CAAB
J	5	CI	CBADD
J	5	CI	CBACD
J	5	CI	CBCBD
J	5	CI	CDACD
J	5	CI	CCAAB
J	5	ECE	BBCBB
J	5	CBBBB	CBBBB
K	6	BJ	BCBADD
K	6	BJ	BCBACD
K	6	BJ	BCBCBD
K	6	BJ	BCDACD
K	6	BJ	BCCAAB
K	6	BJ	BBBCBB
K	6	BJ	BCBBBB
K	6	DBFCC	DBDCCC
K	6	DBFCC	DBDDCC
K	6	DBFCC	DBABCC
K	6	CEAF	CBBADC
K	6	CEAF	CBBADD
K	6	CEAF	CBBAAB
L	7	KA	BCBADDA

L	7	KA	BCBACDA
L	7	KA	BCBCBDA
L	7	KA	BCDACDA
L	7	KA	BCCAABA
L	7	KA	BBBCBBA
L	7	KA	BCBBBBA
L	7	KA	DBDCCCA
L	7	KA	DBDDCCA
L	7	KA	DBABCCA
L	7	KA	CBBADCA
L	7	KA	CBBADDA
L	7	KA	CBBAABA
M	8	LC	BCBADDAC
M	8	LC	BCBACDAC
M	8	LC	BCBCBDAC
M	8	LC	BCDACDAC
M	8	LC	BCCAABAC
M	8	LC	BBBCBBAC
M	8	LC	BCBBBBAC
M	8	LC	DBDCCCAC
M	8	LC	DBDDCCAC
M	8	LC	DBABCCAC
M	8	LC	CBBADCAC
M	8	LC	CBBADDAC
M	8	LC	CBBAABAC
M	8	FBBCAE	DCBBCABB
M	8	FBBCAE	DDBBCABB
M	8	FBBCAE	ABBBCABB
N	10	AAM	AABCBADDAC
N	10	AAM	AABCBACDAC
N	10	AAM	AABCBCBDAC
N	10	AAM	AABCDACDAC
N	10	AAM	AABCCAABAC
N	10	AAM	AABBBCBBAC
N	10	AAM	AABCBBBBAC
N	10	AAM	AADBDCCCAC
N	10	AAM	AADBDDCCAC
N	10	AAM	AADBABCCAC
N	10	AAM	AACBBADCAC
N	10	AAM	AACBBADDAC
N	10	AAM	AACBBAABAC
N	10	AAM	AADCBBCABB
N	10	AAM	AADDBBCABB
N	10	AAM	AAABBBCABB
N	10	AAECGDF	AABBCADDDC
N	10	AAECGDF	AABBCADDDD
N	10	AAECGDF	AABBCADDAB

N	10	AAECGDF	AABBCACDDC
N	10	AAECGDF	AABBCACDDD
N	10	AAECGDF	AABBCACDAB
N	10	AAECGDF	AABBCCBDDC
N	10	AAECGDF	AABBCCBDDD
N	10	AAECGDF	AABBCCBDAB