Journal of Experimental & Theoretical Artificial Intelligence

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/teta20

HLA: a novel hybrid model based on fixed structure and variable structure learning automata

Saber Gholami, Ali Mohammad Saghiri, S. M. Vahidipour & M.R. Meybodi

To cite this article: Saber Gholami, Ali Mohammad Saghiri, S. M. Vahidipour & M.R. Meybodi (2022): HLA: a novel hybrid model based on fixed structure and variable structure learning automata, Journal of Experimental & Theoretical Artificial Intelligence, DOI: 10.1080/0952813X.2021.1960630

To link to this article: https://doi.org/10.1080/0952813X.2021.1960630

Published online: 13 Feb 2022.

JOURNAL OF EXPERIMENTAL & THEORETICAL ARTIFICIAL INTELLIGENCE
https://doi.org/10.1080/0952813X.2021.1960630

Taylor & Francis
Taylor & Francis Group

ARTICLE

HLA: a novel hybrid model based on fixed structure and variable structure learning automata

Saber Gholami[a,b], Ali Mohammad Saghiri[b,c], S. M. Vahidipour[d] and M.R. Meybodi[b]

[a]Department of Computer Science and Software Engineering, Concordia University, Montréal, Canada; [b]Department of Computer Engineering, Amirkabir University, Tehran, Iran; [c]Information and Communication Technology Research Department, Niroo Research Institute, Tehran, Iran; [d]Computer Engineering Department, University of Kashan, Kashan, Iran

ABSTRACT

Learning Automata (LAs) are adaptive decision-making models designed to find an appropriate action in unknown environments. LAs can be classified into two classes: variable structure and fixed structure. To the best of our knowledge, there is no hybrid model based on both of these classes. In this paper, we propose a model that brings together the benefits of both classes of LAs. In the proposed model, called an HLA, the action switching phase of a fixed structure learning automaton is fused with a variable structure learning automaton. Several computer simulations are conducted to study the performance of the proposed model with respect to the total number of rewards and action switching in addition to the convergence rate. The proposed model is compared to both variable structure and fixed structure learning automata, and in most cases, the numerical results demonstrate its superiority. In order to show the applicability of the HLA, a novel adaptive dropout mechanism in deep neural networks was suggested. The results of the simulations show that the proposed mechanism performs better than the simple dropout mechanism with respect to network accuracy.

ARTICLE HISTORY
Received 26 February 2020
Accepted 12 July 2021

KEYWORDS
Learning automata; hybrid learning models; fixed structure learning automata; variable structure learning automata; adaptive dropout

Introduction

Due to the significant growth of the amount of data that is being generated on a daily basis, it is almost impossible to deploy human agents regarding many tasks. Thus, it is reasonable to utilise artificial intelligence as well as machine learning algorithms to decrease the cost and improve efficiency. Machine Learning includes, but is not limited to, three paradigms; supervised learning, unsupervised learning, and reinforcement learning. Learning Automata (LAs) are a well-known family of the latter category. In this study, we focus on LAs with the aim of proposing a new hybrid model that improves the performance of previous models.

As shown in Figure 1, LA is an adaptive decision-making unit that improves its performance by learning how to determine an optimal action out of a set of allowable actions through repeated interactions with a random environment (Narendra & Thathachar, 2012). In other words, a finite number of actions can be performed in a random environment. When a specific action is performed, the environment provides a response that can be either favourable or unfavourable. The objective in the design of the automaton is to determine how the choice of the action at any stage should be guided by the past actions and responses (Narendra & Thathachar, 2012). Besides, the theory of

CONTACT Ali Mohammad Saghiri, saghiri@aut.ac.ir Information and Communication Technology Research Department, Niroo Research Institute, Tehran, Iran; Department of Computer Engineering, Amirkabir University, Tehran, Iran

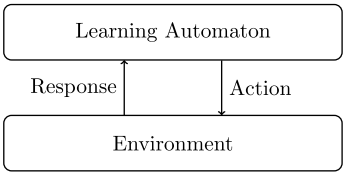

Figure 1. The interaction between Learning Automaton (LA) and Environment

learning automata has been used in a wide range of applications in recent years (Yazidi et al., 2016, 2019; Zhang et al., 2019).

In a general view, Learning Automata can be divided into two classes (Narendra & Thathachar, 2012): Fixed Structure LAs (or FSLAs) and Variable Structure LAs (or VSLAs). In the first class, a state machine is used to find the favourable action among a set of allowable actions, while in VSLAs a probability vector is used to find the favourable action among a set of allowable actions. An updating schema will be used in order to learn how to choose the best action. In this study, we aim to provide a more sophisticated LA based on these two classes.

The main drawback of FSLAs is that they suffer from a weak switching action mechanism, which leads to a low reward value in many situations. To the best of our knowledge, there is no solution for resolving this issue. This study proposes a new hybrid model, created by fusing a fixed structure LA with a variable structure LA to overcome this drawback and combine the benefits of both classes. The main contribution of this paper is its proposed hybrid model based on both FSLA and VSLA to satisfy the following objectives:

- Perform at least as good as both FSLAs and VSLAs;
- Resolve the weak action switching mechanism of FSLAs
- Be robust in terms of changing the updating schema, the number of allowable actions, and other parameters;
- Outperform more sophisticated families of LAs such as pursuit LA;
- Converge quicker than other learning models;
- Be less sensitive to noise; and
- Perform well in real-world applications where many constraints and limitations may exist.

Both FSLA and VSLA models have been used in a wide range of applications in different domains. Since the proposed model does not invest in high computational power, and its structure is generally similar to that of FSLAs and VSLAs, it could be deployed with small modifications in every situation where LAs are useful. Therefore, the proposed model will have an enormous number of applications. In addition, as a particular application of the proposed model, a novel adaptive dropout mechanism in deep learning is suggested to avoid overfitting. The novelty of the proposed model would imply the originality of its application in deep neural networks as well.

Extensive computer simulations were conducted to evaluate the performance of the proposed model. We first examine the efficiency of the model on its own, and then analyse the performance of the adaptive dropout mechanism based on the proposed learning model. The numerical results demonstrate the superiority of our model compared to both FSLAs and VSLAs, with a significant improvement (up to 50%) in terms of the convergence rate. In addition, the proposed model is able to perform impressively in real-world applications, while maintaining its performance in noisy environments.

The rest of this paper is organised as follows: the next section considers some of the basic preliminaries of this work. Section 3 discusses related works, and then Section 4 presents the proposed model and how it is employed in deep neural networks. The performance of the proposed method is evaluated through simulation experiments in Section 5. Finally, we conclude the paper and propose some future works in Section 6.

Preliminaries

In this section, we will discuss the preliminaries of our work. To this aim, we discuss two classes of Learning Automata, namely FSLA, and VSLA.

Fixed Structure Learning Automata (FSLAs)

Different forms of FSLAs are reported in the literature including Tsetlin, $G_{2N,2}$, Ponomarev, Krylov, and so on. In this study, we focused on the Tsetlin family of FSLAs. We first describe $L_{2N,2}$ and then $L_{kN,k}$; which are well-known extensions of the Tsetlin machine.

• $L_{2N,2}$: is an FSLA which has two actions, each of which benefiting from memory of depth N. When action switching is required, the next action, clock-wise, is selected.

• $L_{kN,k}$: An $L_{2N,2}$ automaton with k allowable actions is called $L_{kN,k}$. In both models, receiving reward results in going one step deeper in the memory depth. Furthermore, by getting a penalty from the environment, the automaton goes one step backward in the memory depth. Similarly, when action switching is required, the next action, clock-wise, is selected.

The simplest fixed structure LA is the two-state automaton $L_{2,2}$ which has two states, ϕ_1 and ϕ_2, and also two actions, α_1 and α_2. The automaton accepts a reinforcement signal from a finite set $\{0, 1\}$ and changes its state when receiving an unfavourable response or a penalty ($\beta = 0$). It also remains in the same state when it receives a favourable response or a reward ($\beta = 1$). The structure of an $L_{2,2}$ could be found in Figure 2.

It is obvious that in this model, the state could change roughly since no memory is defined. To tackle this problem, a more sophisticated model, namely $L_{2N,2}$, is proposed which has $2 \times N$ states and 2 actions and attempts to incorporate the past behaviour of the system in its decision for choosing the sequence of actions. It is shown in (Narendra & Thathachar, 2012) that automata with memory normally perform better compared to memory-less ones. Figure 3 represents an $L_{2N,2}$ with $N = 2$.

The described automaton has only two actions. The same concepts may be extended to cases where the automata can perform k actions: $\alpha_1, \alpha_2, \cdots, \alpha_k$. This automaton is called $L_{kN,k}$. The most significant difference between $L_{kN,k}$ and $L_{2N,2}$ happens when switching among k actions is concerned. The state transition graph is shown in Figure 4 for the case where $k = 4$ and $N = 2$. In this automaton, being in state ϕ_i corresponds to choosing action α_i.

An initial state is chosen in Figure 4, for instance, ϕ_1. Afterwards, based on the reward $(\beta = 1)$ or penalty $(\beta = 0)$, the state becomes deeper or weaker, respectively. When the automaton is to switch among different actions, the next action, clock-wise, will be chosen.

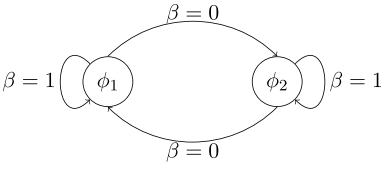

Figure 2. An $L_{2,2}$ automaton

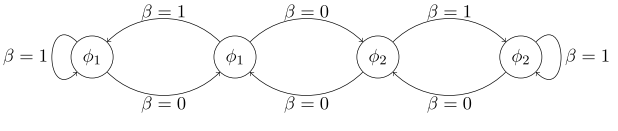

Figure 3. An $L_{2N,2}$ automaton with $N = 2$

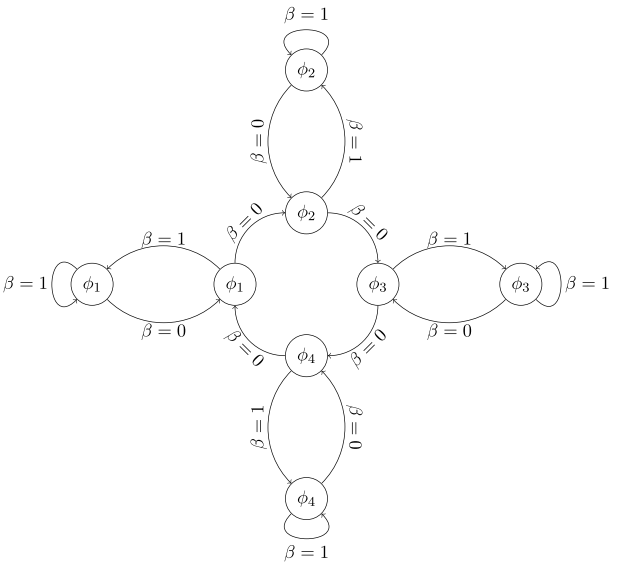

Figure 4. An $L_{kN,k}$ automaton with $k = 4$ and $N = 2$

Variable Structure Learning Automata (VSLAs)

Variable structure learning automata are formulated by a quadruple $< A, \beta, \tau, p(n) >$, where $A = \{\alpha_1, \alpha_2, \cdots, \alpha_k\}$ is the finite set of allowable actions and β is the reinforcement signal. Also, τ is the learning algorithm which is used for updating action probabilities, and $p(n)$ is the probability vector at instance n: $p(n) = [p_1(n), p_2(n), \cdots, p_k(n)]^T$, where $p_j(n)$ is the probability of choosing action j at instance n (Thathachar & Sastry, 2011). The recurrence equation (1) is a linear learning algorithm that the action probability vector is updated when the chosen action is rewarded by the environment ($\beta = 1$):

$$p_j(n+1) = \begin{cases} p_j(n) + \lambda_1(1 - p_j(n)) & \text{if } j{=}i \\ (1 - \lambda_1)p_j(n) & j{\neq}i \end{cases} \tag{1}$$

The probability vector will also be updated using the following equation upon receiving a penalty ($\beta = 0$):

$$p_j(n+1) = \begin{cases} (1 - \lambda_2)p_j(n) & \text{if } j{=}i \\ \frac{\lambda_2}{k-1} + (1 - \lambda_2)p_j(n) & j{\neq}i \end{cases} \tag{2}$$

Variable k is the number allowable actions. λ_1 and λ_2 denote the reward and penalty parameters, respectively. They also represent the amount of increases and decreases in the action probabilities (Thathachar & Sastry, 2011). Different forms of updating schema reported for VSLAs in the literature includes, but is not limited to:

- L_{R-I}: an updating schema in which rewarding an action leads to a change considering rate λ_1 in the probability vector of the LA in the next steps. λ_1 is called the reward parameter $(0 < \lambda_1 < 1, \lambda_2 = 0)$.
- L_{P-I}: an updating schema in which penalising an action leads to a change considering rate λ_2 in the probability vector of the LA in the next steps. λ_2 is called the penalty parameter and $(0 < \lambda_2 < 1, \lambda_1 = 0)$.
- L_{R-P}: an updating schema in which both rewarding and penalising an action lead to an increase (with the rate λ_1) and decrease (with the rate λ_2) in the probability of choosing that specific action in the next steps, respectively $(\lambda_1 = \lambda_2)$.

• $L_{R_\epsilon - P}$: a similar schema with L_{R-P}, but the value of λ_1 is greater than λ_2, $(\lambda_1 >> \lambda_2)$.

Related Works

In this section, the related works of this study are discussed. Since this paper focuses on designing a novel hybrid model based on FSLAs and VSLAs and then suggests a novel dropout mechanism in deep neural networks, this section is organised into two parts as follows: We first discuss the relevant literature of the theory of learning automata and then, the applications of LAs in neural networks are summarised.

On the one hand, learning automata are known as simple decision-making units. Recently, the theory of learning automata has been merged with different fields to create better decision-making units regarding different purposes. Some of these combinations that are described thoroughly in (Rezvanian et al., 2018) are as follows:

• Cellular Learning Automata (CLAs): CLAs invest in designing a team of decision-makers that operates in a fully distributed system. They combine the computational power of cellular automata with the learning capability of learning automata in the unknown environments.

• Distributed Learning Automata (DLAs): DLAs contains a team of decision-makers who try to find an appropriate path in a distributed manner, and they are organized to learn a sequence of decisions in unknown environments.

• Adaptive Petrinet based on learning automata: these models focus on designing a team of learning automata which are organized based on a petrinet. These are powerful models for modelling and simulation, and their capabilities are merged with learning automata theory to present learning mechanisms that can be analysed using petrinet theories.

All of the mentioned approaches and some of those reported in (Rezvanian et al., 2018) invest in designing a learning mechanism obtained from the learning abilities of a team of learning automata. It is obvious that designing a novel and efficient learning automaton has a wide impact on different domains related to the theory of learning automata. It should be emphasised that the problem of non-adaptive action switching in $L_{kN,k}$ has not been addressed yet, and there is no recently reported solution for this problem. Additionally, there is no hybrid model based on both FSLA and VSLA in the literature and the main contribution of our paper is to present the first one.

On the other hand, learning automata theory is widely used in the domain of neural networks from simple perceptron models to deep neural networks. In what follows, some of the state-of-the-art methods in this area are summarised.

As a primary study in this domain, we can refer to (Aguilar, 1993) that tried to study several learning systems such as learning automata, neural networks, and adaptive learning with a particular focus on the first one. The main contribution of this study is to analyse the behaviour of a team of LAs for learning a set of weights. This solution, in comparison with deterministic back-propagation algorithms, results in faster convergence.

In (Meybodi & Beigy, 2002a) a learning automata-based algorithm for parameter adaption in the feed-forward neural network is presented to avoid local optima. The parameters that are tuned using learning automata are learning rate (η), momentum factor (α), and steepness (λ). In (Meybodi & Beigy, 2002b), learning automata-based parameter adaption algorithms are given to avoid local optima and approach to the global optimum. In (Sudareshan & Condarcure, 1998) learning automata theory is applied to design a training mechanism in recurrent neural networks, where the automaton's actions are used to increase and decrease the network parameters. The aim of (Beigy et al., 2002)is to deploy fixed structure learning automata to tune the parameters of back propagation algorithms in feed-forward neural networks. In the next paragraph, we focus on the usage of learning automata in complex neural networks such as convolutional neural networks and deep neural networks.

In (Guo et al., 2018), Learning automata Competition Unit (LCU) is defined, in which there exist several convolutional kernels and an LA. During training, the LA assists to select the most well-trained kernel. This kernel produces the output of the corresponding LCU to the connected LCU(s) in the next layer. Indeed, the selected kernels through competition can make the training process more

efficient. In (Guo et al., 2017), considering the difficulties of the training deep models, i.e. a large number of parameters and multiplication operations, a method of gradually pruning the weakly connected weights is proposed. To this end, learning automata is used to find the weakly connected weights in the neural networks. This method changes the initially fully connected neural networks into a more effective and sparsely connected architecture during the training phase. In (Guo et al., 2019), a deep neural network is equipped with learning automata to propose an effective incremental training method. To this end, a connection between two neurons has two states: 'activated' or 'deactivated'. During the incremental training stages of this network, the learning automata make decisions about the states of links lead to better training performance on the incremental learning stage. In (Feng et al., 2018), using learning automata, a scheduling method is provided to compress the number of convolutional kernels. This method can effectively compress the number of convolutional kernels at the expense of losing weak classification accuracy.

In this paper, we propose a hybrid model based on the theory of both VSLA and FSLA. More specifically, the $L_{kN,k}$ model of FSLA will be fused with the VSLA. As it was previously mentioned, $L_{kN,k}$ uses a clock-wise action switching mechanism in some situations which leads to low performance. We will solve this problem by utilising the decisions of a VSLA instead of the clock-wise action switching mechanism. To the best of our knowledge, this type of combination is unique and is not similar to any learning automata reported in the literature till now. Besides, a novel dropout mechanism as an application of the learning automata in deep neural networks will be suggested in this paper that is based on HLA. Note that there is no adaptive dropout mechanism based on learning automata in the literature.

The Proposed Method

We present our hybrid model, called the HLA, in this section. The HLA is based on variable structure learning automata and a $L_{kN,k}$ model of fixed structure learning automata. At the end of this section, an application of the proposed model in deep neural network is given. In this application, the HLA is used to design an adaptive dropout method.

Proposed Model

The HLA takes four parameters for its construction (k: its number of actions, N: its memory depth, λ_1: the reward parameter, and λ_2: the penalty parameter). We considered the P-model for modelling the environment of the HLA, in which the automaton may receive a value of β which is either 0 or 1 as the input of the update algorithm. The structure of this model is given in Figure 5. This model has three units as follows:

• FSLA: This unit implements a fixed structure learning automaton based on $L_{kN,k}$ and takes two parameters: k and N. In addition to two common functions of FSLA, namely *action_selection*() and *update*(β), another function is considered in this study: *is_on_edge*() returns true if the FSLA is on the edges and is ready for changing the action when receiving a penalty. Otherwise, it returns false.

• VSLA: This unit implements a variable structure learning automaton based on linear learning algorithms. VSLA gets three parameters; k,λ_1, and λ_2. This unit also has two functions; *action_selection*() and *update*(β) for decision-making and updating its probability vector, respectively.

• Fusion manager: This unit is in charge of managing decision-making and updating in a novel way. It implements the *action_selection*() and *update*(β) functions of the HLA as given in Figure 6 and Figure 7. These functions will be described in detail.

In the pseudo-code of Figure 6, two variables are defined to help the fusion manager unit make its decision. The first variable, called *mode*, stores the type of LA that should be responsible for the next decision. For instance, if $mode = V$, it means that VSLA should decide on the next action. Additionally, the second variable, namely *turn*, stores the type of automata that is responsible for

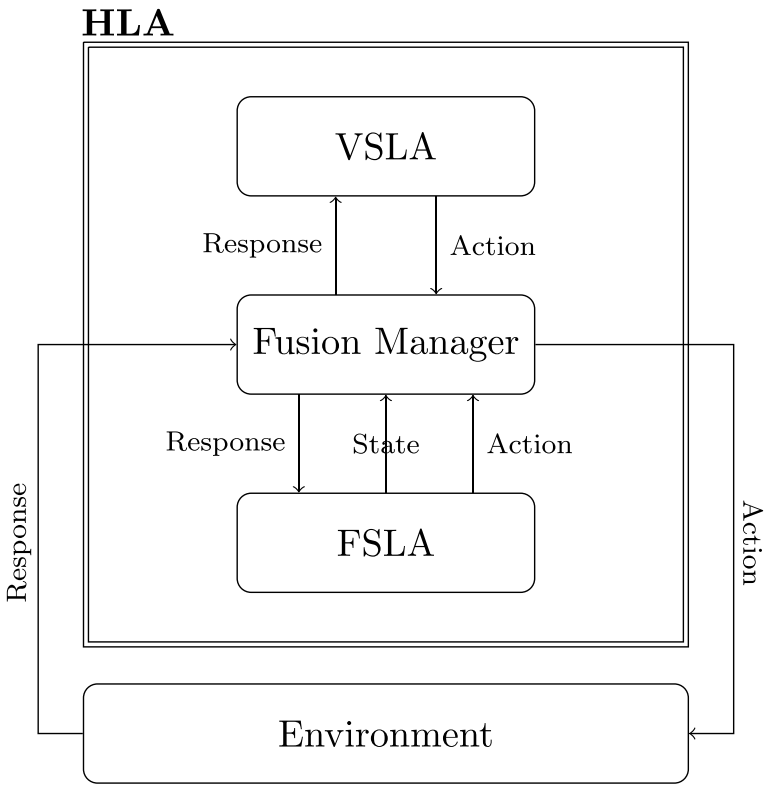

Figure 5. The structure of HLA

Algorithm 1 *action_selection*()

Notation: VSLA denotes a variable structure learning automaton,
FSLA denotes the fixed structure learning automaton with $L_{kN,k}$ learning algorithm,
mode is a variable that is initialized to V and stores the unit responsible for the next action,
turn is a variable that is initialized to V and monitors the type of LA that selected the last action,
A_l denotes the action chosen by FSLA or VSLA,
β denotes the reinforcement signal.

```
begin
    if mode=='V' then
        A_l ←VSLA.action_selection()
        turn ← 'V'
        mode ← 'F'
    end
    else
        A_l ← FSLA.action_selection()
        turn ← 'F'
    end
    return A_l
end
```

Figure 6. Pseudo-code for the action_selection function of HLA

Algorithm 2 *update*(β)

Notation: VSLA denotes a variable structure learning automaton,
FSLA denotes the fixed structure learning automaton with $L_{kN,k}$ learning algorithm,
mode stores the responsible unit for the next action,
turn keeps track of the type of LA that selected the last action,
β denotes the reinforcement signal.

```
begin
    if β==1 then
        if turn=='V' then
            VSLA.update(1)
            mode ← 'F'
        end
        else if turn=='F' then
            FSLA.update(1)
        end
    end
    else if β==0 then
        if turn == 'V' then
            VSLA.update(0)
            mode ← 'V'
        end
        else if turn=='F' then
            if FSLA.is_on_edge() then
                VSLA.update(0)
                mode ← 'V'
            end
            else
                FSLA.update(0)
            end
        end
    end
end
```

Figure 7. Pseudo code for the *update* function of the HLA

the last chosen action. For example, if *turn* = *F*, it means that the last action has been chosen by the FSLA, and so the FSLA should either be rewarded or penalised based on the response of the environment.

HLA is executed within two different functions. As can be seen in Figure 6, the purpose of the first function is to select an action. To this aim:

• If the next action must be chosen by the VSLA (*mode* == *V*), it chooses an action. Afterwards, *turn* is set to *V*, as the embedded VSLA is responsible for choosing the last action. We set *mode* to *F*, because from now on, the FSLA must select the next actions, unless it is told to do otherwise.

• But, if the FSLA should decide on the next action (or *mode* == *F*), the respective function of the FSLA is invoked. Subsequently, *turn* is set to *F*, as the FSLA has made the decision on the last chosen action.

In either of these two circumstances, the chosen action is returned to the environment. On the other hand, the aim of the second function is to update the appropriate automata (Figure 7):

• If the HLA is rewarded by the environment ($\beta = 1$), the LA that has chosen the last action must receive this reward; thus, two scenarios are possible:

° If the VSLA has chosen the last action, its probability vector has to be updated in favor of the chosen action. Consequently, the FSLA will be in charge of choosing the next actions since the VSLA has performed its responsibility.

° Otherwise, if the FSLA has chosen the last action, it must be rewarded and it will remain responsible for the next action since the FSLA is doing well to this point.

• However, if the HLA is penalised by the environment $(\beta = 0)$, again, two scenarios are possible:

° If the VSLA has chosen the last action, it should be penalized. Also, the VSLA remains responsible for choosing the next action since it could not find the best action so far, and it has to keep trying until its probability vector converges.

° But, if the FSLA has chosen the last action, the HLA decides on penalizing one of the automata, based on the depth of the FSLA. If the FSLA is on its edges, it means that it should change the action and choose a new one. Also, since the last action was not good enough, the VSLA is penalized for guiding the FSLA through the selected action. The VSLA must also choose the next action using its updated probability vector. However, if the FSLA is not on its edges, it is penalized by the environment but remains responsible for choosing the next action as long as it is in memory depth of an action.

To explain the working procedure of the HLA, an example is described in Figure 8. In this figure, $< \alpha_i, j >$ corresponds to action i when being in memory depth j. Suppose an HLA with four allowable actions ($\alpha_1, \alpha_2, \alpha_3,$ and α_4), where α_3 is the favourable one, and the memory depth is $N = 2$. At the initial stage, an action must be chosen with respect to the probability vector. Let us say α_1 is selected with a depth of 1 (Figure 8a). If this action does not satisfy the environment, α_1 will be penalised. The HLA's probability vector will then be updated (Figure 8b).

Imagine that this time the probability vector chooses α_3. So, the HLA goes to the first stage of action 3 (Figure 8c). Now if this action receives a reward from the environment, two things will happen. First, the HLA's embedded FSLA goes one step deeper in the memory depth. Next, the probability vector will be updated in favour of action 3 because it has received a reward from the environment. As shown in Figure 8d, not only is the HLA in the depth of action α_3, but its embedded probability vector works in favour of α_3. Thus, in the case of a required switching action, the favourable action is more likely to be selected by the HLA.

The following remarks can be considered for the HLA:

• An HLA can be converted to a pure chance automaton when $N = 0, \lambda_1 = 0,$ and $\lambda_2 = 0$.

• An HLA can be converted to a variable structure learning automaton when $N = 0$.

• An HLA can be converted to $L_{kN,k}$ with random initialisation when $\lambda_1 = 0$, and $\lambda_2 = 0$.

• In an HLA, an $L_{kN,k}$ is used to define a memory for a variable structure learning automaton. It should be noted that the $L_{kN,k}$ automaton performs well when the number of actions is high and the environment is dynamic, but the variable structure learning automata perform worse in the same situation (Narendra & Thathachar, 2012). It seems that an HLA would be able to perform well when the number of actions is high and the environment is dynamic, as it inherits the features of the $L_{kN,k}$ automaton.

• In an HLA, a variable structure learning automaton is used to conduct the action initialisation process for $L_{kN,k}$. Therefore, in the long run, an HLA is able to avoid local optima as well as learn a good policy for the action initialisation process because it inherits the characteristics of variable structure learning automata.

• Consequently, since an HLA inherits the characteristics of both FSLA and VSLA, we expect that it would perform impressively in dynamic environments with many allowable actions and avoid local optima. These claims will be verified in our experiments.

An Application of the Proposed Model

In this part, an HLA is used to design a novel adaptive dropout method in deep neural networks. As mentioned, the theory of learning automata has not been used to design a dropout algorithm in the literature. The dropout method and related concepts are introduced next, followed by a novel adaptive dropout algorithm based on the HLA.

Overfitting is a challenging problem in deep neural networks, as it is more serious and difficult in large networks. The dropout method has been introduced to address this problem (Srivastava et al., 2014), in which some neurons and their connections are randomly dropped from the network during the training procedure. This algorithm prevents units from co-adapting excessively. In this mechanism, the choice of which units to be dropped is random, and a unit is retained with a fixed probability independent of other units, usually from range $[0.5, 1]$. The fixed value is assigned to each unit. More importantly, many experiments are required to choose the optimal value for a wide range of networks and tasks. In practice, during training, the dropout method samples from an exponential number of several 'thinned' networks. When testing, it is easy to approximate the effect of averaging the predictions of these thinned networks by simply using a single un-thinned network that has appropriate weights. The implementation of (Srivastava et al., 2014) is available online,[1] and we use it to suggest a new dropout algorithm based on the HLA.

In the proposed method, the network is equipped with a three-action HLA. The action set of each learning automaton is {'increase probability', 'don't change probability', and 'decrease probability'}. The step size of decreasing or increasing the value of probability is denoted by the *step* parameter. In addition, the decision of learning automata to increase and decrease the value of probability to a region outside of upper-bound and lower-bound will be ignored.

The learning of an HLA is fused within the learning of the dropout method. Before learning, training data is divided into several mini-batch data. For all training cases in a mini-batch, a thinned network is sampled by dropping out units (HLA is used to sample the thinned network in our proposed method). Indeed, this thinned network is used in forward and back-propagation for those training cases. We assume that only one HLA exists by which one dropout probability is adjusted. This probability is used to choose dropping units during the learning of a mini-batch. The gradients for each parameter are averaged over the training cases in each mini-batch. After back-propagation, the loss value of the thinned network per mini-batch is calculated, and will be used to generate the reinforcement signal for the HLA. For example, if the current loss value becomes the minimum value of all the calculated loss values thus far, the HLA is rewarded. The pseudo-code of the proposed algorithm is illustrated in Figure 9.

$$P \longrightarrow 0.5$$

$$p \longrightarrow p + step$$

$$P \longrightarrow 1$$

$$minloss \longrightarrow loss$$

$$HLA.update(1)$$

$$HLA.update(0)$$

$$iter \longrightarrow iter + 1$$

Experimental Results

To show the efficiency of the proposed model, we have conducted several computer simulations. In this section, each of which will be discussed in detail. We first begin with evaluation metrics, and afterwards, the numerical results of our study are discussed.

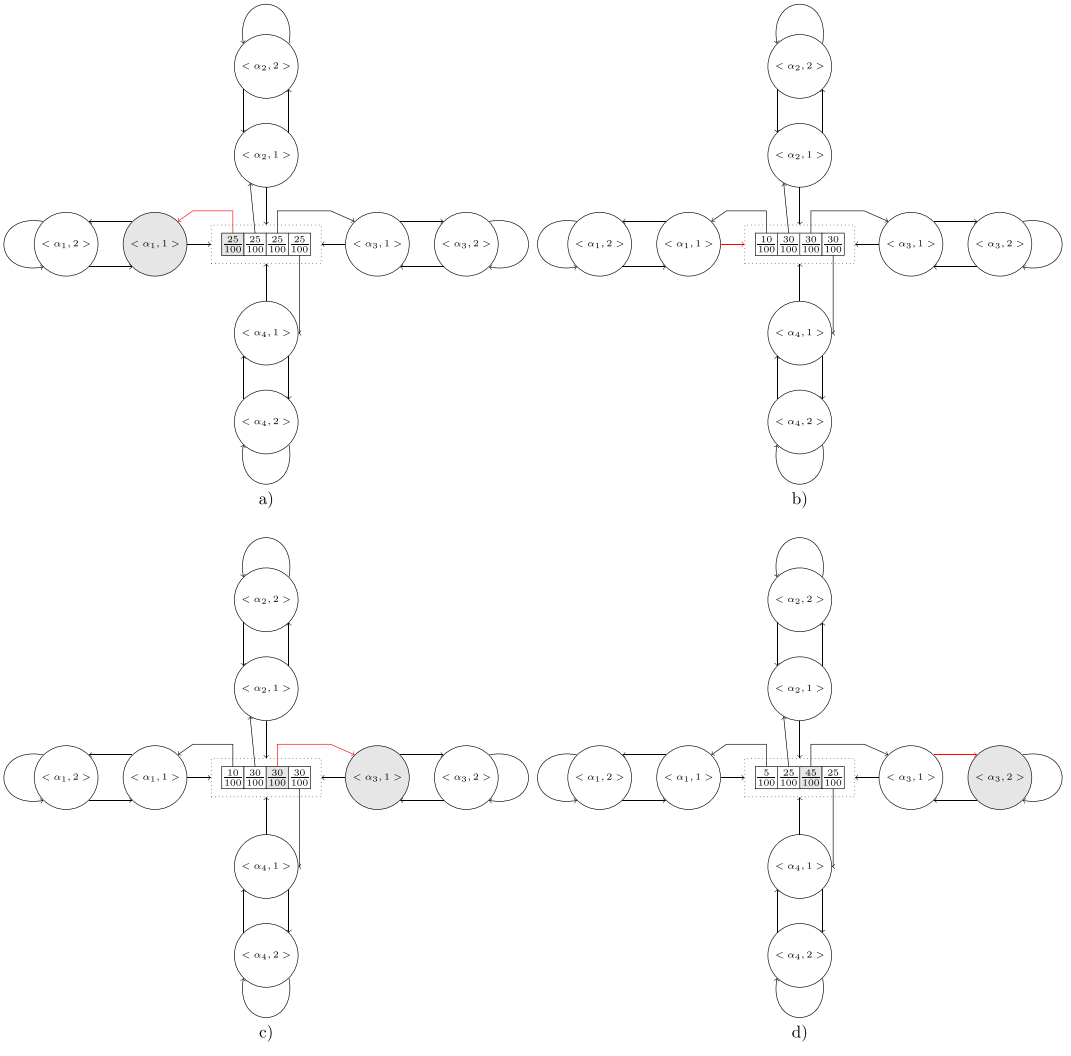

Figure 8. (a-d) An example of HLA

Simulation setup and evaluation metrics

All the simulations of this study are performed on an Intel Core i7 CPU with 2.7 GHz frequency. We also used Python for the simulation due to its reputation and also great speed and simplicity. Also, HLA's performance will be evaluated with respect to the following criteria:

Total Number of Rewards (TNR)

The automaton could be rewarded or penalised based on the feedback of the environment. In iteration *i*, if the automaton is rewarded by the environment ($\beta_i = 1$), *TNR* will be added by 1. So, TNR_i is the summation of the number of rewards in previous iterations:

$$TNR_i = \sum_{j \leq i} \beta_j \tag{3}$$

Algorithm 3 *train_step*()

Notation: HLA denotes a 3-action hybrid learning automaton is a variable for saving the probability of dropout algorithm,
step is a variable for saving the value for decreasing and increasing,
loss is a variable for saving the loss of neural network after a training step,
minloss is a variable for saving the minimum loss of neural network after training steps,
MaxIter defines the maximum number of iteration.

```
begin
    iter = 1, minloss = ∞
    Initialize  p = 0.8
    while iter < MaxIter do
        A mini-batch data is formed form the training data
        current ← HLA.action_selection()
        if current == "decrease probability" then
            p ← p − step
            if P < 0.5 then
                P ← 0.5
            end
        end
        if current == "increase probability" then
            p ← p + step
            if P ≥ 1 then
                P ← 1
            end
        end
        A thinned network is sampled by dropping out units. The probability value p
         is used to determine the dropped out units.
        Train thinned neural network using a mini-batch of training data.
        Save loss obtained from trained neural network
        if loss < minloss then
            minloss ← loss
            HLA.update(1)
        end
        else
            HLA.update(0)
        end
        iter ← iter + 1
    end
end
```

Figure 9. Pseudo-code for the HLA training step

Total Number of Action Switching (TNAS)

The action switching of LAs is costly in some environments. Therefore, we defined a metric to consider this issue. If the automaton changes its decision in iteration *i*, a variable, called s_i, will be set to 1, otherwise 0. Therefore, the total number of times that each automaton has changed its decision until iteration *i* is stored in $TNAS_i$:

$$TNAS_i = \sum_{j \le i} s_j \tag{4}$$

Probability of Penalty (PP)

The probability that the learning automaton has been penalised by the environment is calculated as follows:

$$PP = \frac{\text{number of times that the automaton is penalized by the environment}}{\text{total number of action selection}} \tag{5}$$

Clearly, a smaller value for *PP* is desired for a better-performing LA.

Probability of choosing the favourable action $(p(\alpha_i))$

In cases where there is more than one allowable action to choose and one of them is the favourable one, the probability of choosing that promising action must be calculated. Imagine among a set of actions $\{\alpha_1, \ldots, \alpha_k\}$, α_i is the favourable action for the environment. As a result, the total number of times that α_i is chosen divided by the total number of times that any action has been chosen represents the probability of choosing that favourable action:

$$p(\alpha_i) = \frac{\text{number of times that } \alpha_i \text{ has been selected}}{\text{total number of action selection}} \tag{6}$$

Convergence (C_ε)

An important aspect of evaluating the performance of a learning agent is convergence. We define convergence as the number of iterations needed for a learning automaton to choose the favourable action in at least $1 - \varepsilon$ fraction of times. For instance, an LA satisfies a convergence rate of $\varepsilon = 0.01$ in j iterations, if it chooses the favourable action in at least $1 - 0.01 = 99\%$ of the times in iteration j and after that, and it is denoted by $C_\varepsilon = j$.

Simulation Results

To study the efficiency of the proposed model, several experiments have been conducted. First, the superiority of the model in comparison with pure chance automata is evaluated to test the learning capability of HLA. Afterwards, the benefits of HLA compared to FSLA and VSLA are explored under multiple circumstances, and then, HLA is compared with pursuit LA which is a more sophisticated model. We also designed two more experiments to study the convergence rate of the proposed model as well as its sensitivity to the noise. After evaluating the learning model, the last experiment is dedicated to evaluating the adaptive dropout mechanism based on HLA in deep neural networks. In the following, these experiments are discussed.

Experiment 1: Learning Capability

This experiment is conducted to study the learning capability of HLA. To this aim, it should be compared with a Pure Chance Automaton (PCA) which chooses an action randomly in each iteration. To illustrate, one may consider PCA as a version of HLA in which both reward and penalty parameters, λ_1 and λ_2, are set to 0, and the value of parameter N is set to 1. In this experiment, in each iteration and based on the chosen action, the automaton will be rewarded by the probabilities that are shown in Table 1 (Table 1 gives information about three environments, namely Ex1.1, Ex1.2, and Ex1.3).

For instance, in Ex1.1, the automaton will be rewarded in 10% of times if it chooses α_1, and it also will be rewarded in 90% of times if it chooses α_2. Therefore, the anticipation for HLA is to learn to choose α_2 most of the time since it has a higher chance of being rewarded from the environment. The results of this experiment in terms of *TNR* and *TNAS* are shown in Figure 10. For evaluating the effect of N in HLA, we compared the results of pure chance automata in the mentioned environment with HLA of $N = 1, 4$, and 6. As can be seen in Figure 10, the HLA outperforms pure chance automaton in terms of more *TNR* and less *TNAS*, having a wider margin in Ex1.1 and Ex1.2 compared to Ex1.3. This is mostly because of the nature of the defined environments. To illustrate, in Ex1.3, HLA

Table 1. The setup of experiment 1

	Probability of being rewarded		
Chosen action	Ex 1.1	Ex 1.2	Ex 1.3
α_1	0.1	0.3	0.5
α_2	0.9	0.7	0.5

does not choose an action with confidence, since the probability of getting a reward is equal for both actions.

According to the results of this experiment that are given in Figure 10, we may conclude that the proposed model (HLA) is better than PCA in terms of more *TNR* and less *TNAS*. Consequently, HLA has the capability of learning, obviously.

Besides, the greater the N, the more efficient HLA performs. The underlying reason is that with higher N, HLA does not change its decision roughly. So, fewer *TNAS* and more *TNR*are expected. However, it is shown in (Narendra & Thathachar, 2012) that the behaviour of the fixed structure learning automata with a small value of N, like 4 or 5, is fairly the same with an automaton with infinite memory. Consequently, we assume N = 5 in the rest of this study.

Experiment 2: Comparing with VSLA and the effect of updating schema

The proposed model should be compared with variable structure and fixed structure learning automata (experiments 2,3, and 4). The motivation is to analyse the effectiveness of the HLA model in comparison with the two mentioned models to see whether it has the benefits of both models or not. To this aim, similar experiments with Ex1 are designed. We try to measure the effectiveness of the model in an environment when the probability of being rewarded by choosing one specific action is 80% (favourable action). Other actions, however, have less chance of getting a reward (20% altogether). Since experiment 2–6 share the same general environment, we give the details for that in Table 2 for an arbitrary value of k.

The purpose of Ex2 is to compare HLA with VSLA. For this experiment, a random environment, which is described in Table 2, is considered with $k = 5$. Furthermore, N or the memory depth of HLA is set to 5. Besides, we used 5 different updating schemes to evaluate the robustness of HLA and VSLA to the changes in learning parameters. To clarify, in P-model, both learning parameters λ_1 and λ_2 are set to zero, while in L_{R-I} model, λ_1 is a small constant (0.1 and 0.01), as opposed to λ_2 which is set to 0. In L_{P-I} model, the opposite situation happens where λ_1 is 0 and λ_2 is a small constant of either 0.1 or 0.01. Furthermore, the aim of L_{R-P} model is to evaluate the performance of the automaton in cases where the penalty and reward parameters are equal (we considered 0.1 and 0.01). Lastly, in $L_{R_\epsilon - p}$ model, the value of λ_1 is greater than λ_2 to simulate the situations when rewarding is worthier than penalising. We considered two cases for $L_{R_\epsilon - p} : \lambda_1 = 0.1, \lambda_2 = 0.01$ and

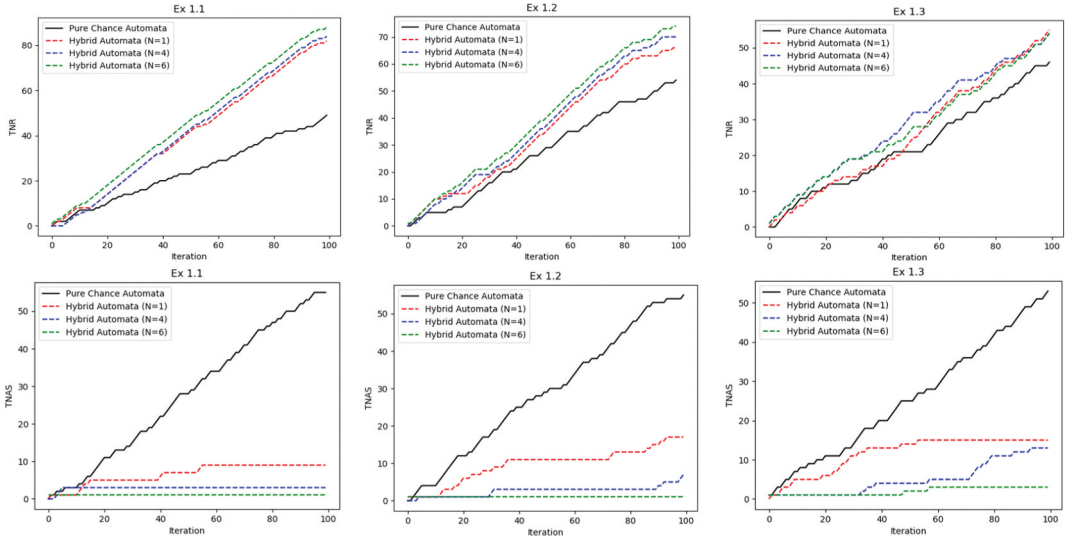

Figure 10. Experimental results of Ex1

Table 2. The setup of experiment 2–6

Number of actions	Probability of being rewarded	
k	$\alpha_1 = 0.8$	$\alpha_2, \cdots, \alpha_k$: each $\frac{0.2}{k-1}$

$\lambda_1 = 0.01, \lambda_2 = 0.001$. All in all, nine different experiments are designed with the mentioned updating schemes which are shown in Table 3.

The final results of Ex2 in Table 4 reveals many interesting points. First, HLA outperforms VSLA in all cases in terms of greater *TNR* with a wide margin. It means that HLA learns to choose the favourable action regardless of the updating scheme. As a result, we claim that HLA is more robust in terms of changing the learning parameters λ_1 and λ_2 in comparison with VSLA. Second, HLA demands fewer *TNAS* compared to VSLA. The underlying reason is that the embedded VSLA in the HLA learns how to choose the next action when the embedded FSLA needs an action switching. Consequently, even when the embedded FSLA of HLA tries to switch the chosen action after several times of being penalised, the embedded VSLA chooses the best action for the next steps, which results in fewer *TNAS*.

Moreover, as it is shown in Figure 11, in HLA, the probability of choosing the favourable action, α_1, is greater compared with that of VSLA in all experiments (left-hand side pictures in Figure 11). Therefore, HLA is less likely to receive a penalty from the environment (right-hand side pictures in Figure 11). Thus, it is claimed that HLA chooses the favourable action regardless of the updating schema, as opposed to VSLA which is heavily dependent on λ_1 and λ_2.

Experiment 3: Comparing with FSLA and the effect of memory depth

The purpose of Ex3 is to compare HLA with FSLA under the same circumstances as Ex2 (see Table 2). Here, the ultimate objective is to analyse the effect of memory depth, N, in both models. We considered four different values of $N = 1, 3, 5, 10$ for analysing the performance of each model with different values of memory depth. Besides, L_{R-I} is used for HLA as the updating schema with $\lambda_1 = 0.1$ and $\lambda_2 = 0$. Moreover, the number of allowable actions is set to $k = 5$. The details of Ex3 is shown in Table 5.

The experimental results of Ex3 are shown in Table 6 and Figure 12. As can be seen, HLA performs impressively even with small values of N, dissimilar with FSLA which demands larger values of N. It means that HLA is robust in terms of changing memory depth which is a great benefit of this model. The reason is that when HLA is constructed with small values of N, like 1, the embedded VSLA in the HLA learns to choose the favourable action. Subsequently, HLA will receive less penalty from the environment as it is capable of choosing the favourable action most of the time (note a high probability of penalty for an FSLA with $N = 1$). Moreover, HLA outperforms FSLA in terms of more *TNR* and fewer *TNAS* in most cases which shows the capability of this model in being rewarded in non-deterministic environments.

Table 3. The setup of Ex2

	P	L_{R-I}		L_{P-I}		L_{R-P}		$L_{R_\epsilon-p}$	
	Ex2.1	Ex2.2	Ex2.3	Ex2.4	Ex2.5	Ex2.6	Ex2.7	Ex2.8	Ex2.9
k	5	5	5	5	5	5	5	5	5
N	5	5	5	5	5	5	5	5	5
λ_1	0	0.1	0.01	0	0	0.1	0.01	0.1	0.01
λ_2	0	0	0	0.1	0.01	0.1	0.01	0.01	0.001

Table 4. Experimental results of Ex2; Comparing HLA and VSLA in terms of *TNR* and *TNAS*

Model	Ex2.1		Ex2.2		Ex2.3		Ex2.4		Ex2.5		Ex2.6		Ex2.7		Ex2.8		Ex2.9	
	TNR	TNAS	TNR	TNAS	TNR	TNAS	TNR	TNAS	TNR	TNAS	TNR	TNAS	TNR	TNAS	TNR	TNAS	TNR	TNAS
HLA	**804**	**3**	**791**	**2**	**768**	**9**	**787**	**14**	**779**	**16**	**788**	**3**	**783**	**6**	**783**	**11**	**776**	**18**
VSLA	195	787	768	36	658	257	221	823	236	782	422	649	357	720	742	103	632	286

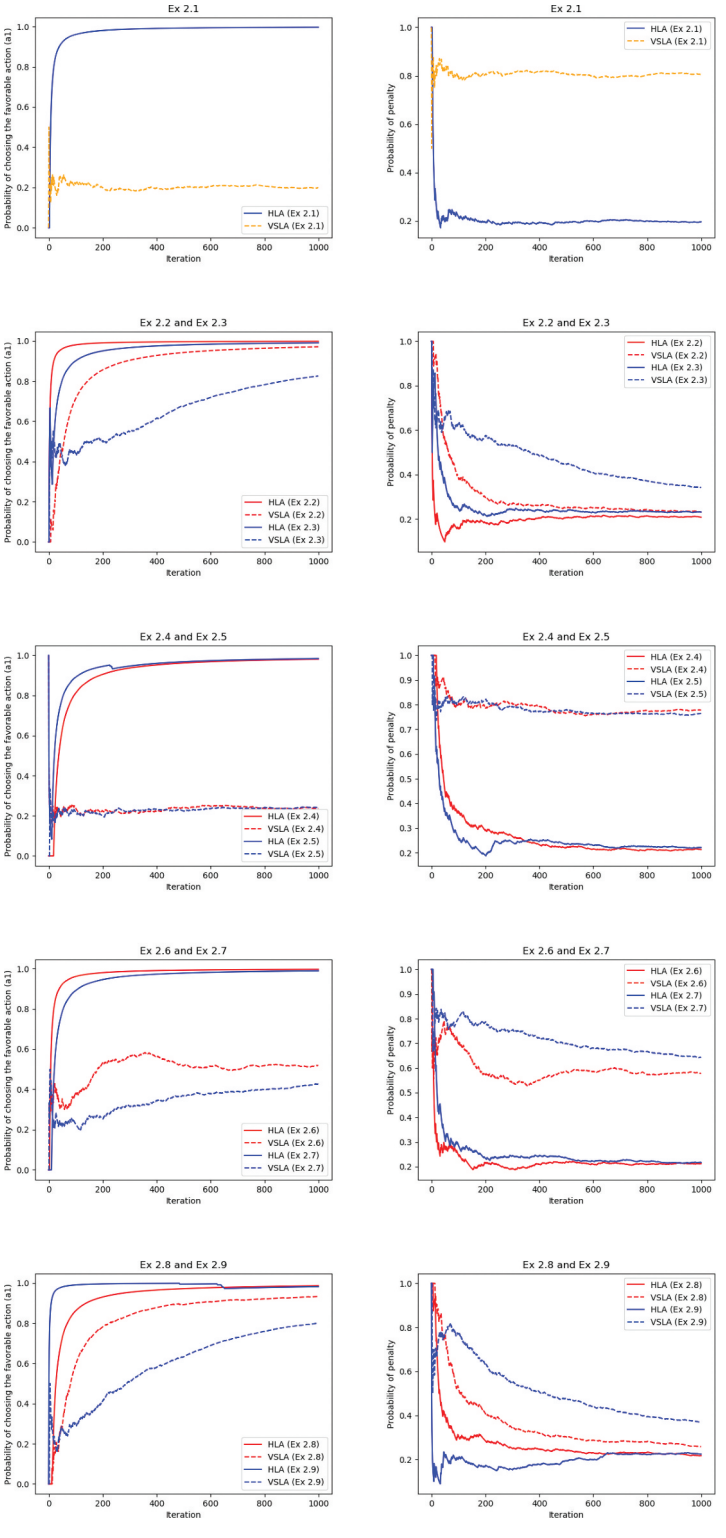

Figure 11. Experimental results of Ex2: the probability of choosing the favourable action, and probability of penalty in HLA and VSLA with different updating schemes

Table 5. The setup of Ex3

	Ex3.1	Ex3.2	Ex3.3	Ex3.4
k	5	5	5	5
N	1	3	5	10
λ_1	0.1	0.1	0.1	0.1
λ_2	0	0	0	0

Table 6. Experimental results of Ex3; Comparing HLA with FSLA in terms of *TNR* and *TNAS*

Model	Ex3.1		Ex3.2		Ex3.3		Ex3.4	
	TNR	TNAS	TNR	TNAS	TNR	TNAS	TNR	TNAS
HLA	**765**	**24**	**781**	**29**	**816**	**6**	**795**	**2**
FSLA	413	400	774	**29**	812	7	791	5

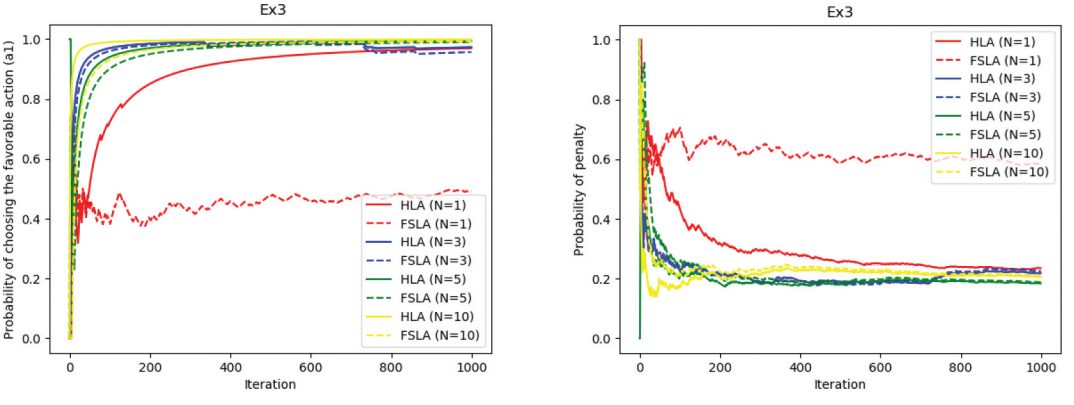

Figure 12. Experimental results of Ex3: the probability of choosing the favourable action, and probability of penalty in HLA and FSLA with different memories

5.2.4. Experiment 4: Comparing with both VSLA and FSLA and the effect of number of allowable actions

The aim of Ex4 is to compare HLA with FSLA and VSLA simultaneously with respect to different values of k (number of allowable actions). Therefore, four experiments are designed in which N is set to 5 and L_{R-I} is being used as the updating schema. In these experiments,four different values of $k = 5, 10, 15, 20$ are considered to investigate the capability of each model in the environments when many actions are allowable. The setup of Ex4 is shown in Table 7.

As can be seen in Figure 13 and Table 8, the final results of Ex4 reveal many interesting points. Firstly, with higher values of k, it is more difficult for all types of learning automata to find the favourable action. So, the slope of $p(\alpha_1)$ is smoother for greater values of k in Figure 13. Thus, HLA has a smaller probability of receiving a penalty from the environment. Furthermore, in all cases, HLA outperforms both FSLA and VSLA in terms of more *TNR* which enables HLA to receive reward most of the time even when the number of actions, k, is high. Also, *TNAS* grows considerably when a higher number of allowable actions are considered in FSLA and VSLA. The underlying reason is that in both models, decision-making becomes tougher when too many actions are acceptable. So, those models have to switch among actions many times to find the favourable one eventually. In HLA, however, *TNAS* is promising in most cases since the embedded VSLA in HLA learns how to choose the next action when it comes to action switching.

Table 7. The setup of Ex4 and 5

	Ex4.1	Ex4.2	Ex4.3	Ex4.4
k	5	10	15	20
N	5	5	5	5
λ_1	0.1	0.1	0.1	0.1
λ_2	0	0	0	0

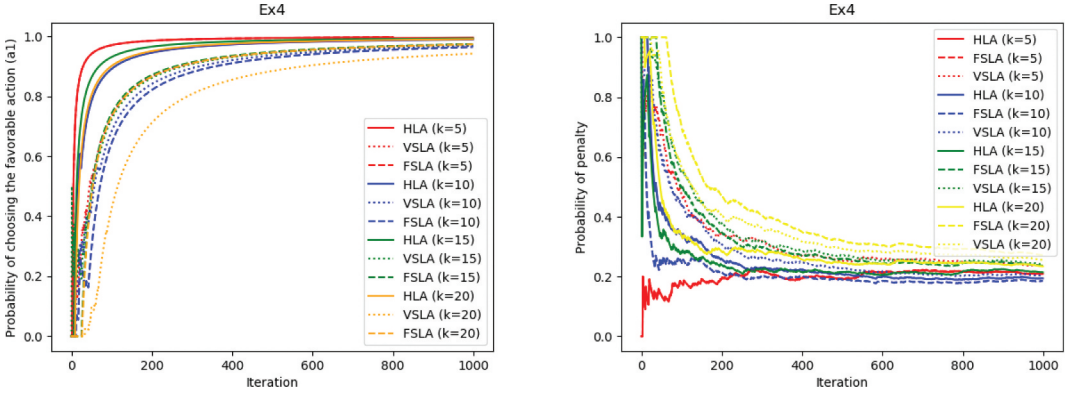

Figure 13. Experimental results of Ex4: the probability of choosing the favourable action, and probability of penalty in HLA, FSLA, and VSLA with different numbers of allowable actions

Table 8. Experimental results of Ex4; Comparing HLA with FSLA and VSLA in terms of *TNR* and *TNAS*

Model	Ex4.1		Ex4.2		Ex4.3		Ex4.4	
	TNR	TNAS	TNR	TNAS	TNR	TNAS	TNR	TNAS
HLA	**781**	**6**	**790**	**12**	**803**	**8**	**792**	**10**
FSLA	779	**6**	772	33	789	22	775	26
VSLA	766	38	774	42	786	36	750	61

5.2.5 Experiment 5: Comparing with Pursuit LA

This experiment aims to provide a comparison of the proposed model with a more sophisticated LA, namely Pursuit LA (PLA). This will be done when the number of allowable actions increases from 5 to 20, while L_{R-I} has been used as the updating schema of both HLA and PLA with the values of $\lambda_1 = 0.1$ and $\lambda_2 = 0$. The memory depth of HLA is also set to 5. Table 2 gives the general setup of the environment for Ex5 as well. The details of this experiment are similar to that of Ex4 and could be found in Table 7.

PLA is a type of LA that is based on VSLA with a small modification. It stores the number of times for which each action has been selected so far (we call it n vector) and also keeps track of the number of rewards that each action receives (called z vector). The ratio of the two mentioned criterion represents a 'goodness' score for that particular action: $d = z/n$. The best (or the *dominant*) action is the one that has the highest value of d vector.

The main difference between VSLA and PLA is that after the action selection phase, the *dominant* action is responsible for the penalty or the reward that is provided by the environment, regardless of the fact that it has actually been selected or not. Thus, in each iteration, the probability vector of the dominant action will be modified only. The values of three vectors, d, z, and n will be changed for the action that was actually selected in that particular iteration, however. It means that, ultimately, the *dominant* action is responsible for the reinforcement signal provided by the environment as long as no other action dominants that action. The results of Ex5 are shown in Figure 14 in terms of $P(\alpha_i)$ and *PP*. Furthermore, the values of *TNR* and *TNAS* for this experiment are also given in Table 9.

The left line graph of Figure 14 shows the superiority of HLA compared to PLA in terms of $P(a_1)$, having a wide margin when the number of allowable actions increases to more than 10. The right-hand side picture also demonstrates that although PLA is able to achieve a small probability of penalty in the long run, it cannot perform as well as HLA at the beginning. The reason is that PLA does not change its decision roughly, even if the selected action is not the favourable action. So, as long as the *dominant* and the favourable action are not the same, PLA will be penalised by the environment. This will lead to a high chance of receiving a penalty until the *dominant* and the favourable action do not match for PLA. The proposed model, HLA, does not treat any action differently and only looks for the favourable one. As soon as it finds that action, it uses its memory to store that action. This leads to a small probability of penalty, especially when the number of allowable actions is not too high.

The main advantage of PLA compared to HLA could be found in Table 9. Clearly, PLA demands a notably fewer number of action switching compared to HLA. This is due to the fact that PLA insists on its current action until another, hopefully favourable, action dominants its d vector, in spite of HLA which can change its decision and look for the favourable action until it goes down to the memory depth of the favourable action and also reaches an acceptable convergence in p vector. Therefore, HLA might switch its action a couple of times in the beginning. Looking at Table 9 it is observed that HLA will be rewarded more in comparison with PLA (a 2–10% increase). Based on the results of this table we argue that the higher number of *TNAS* for HLA is useful since it makes the proposed model achieve a higher *TNR*.

All in all, the experimental results of Ex5 show that HLA is able to even outperform the sophisticated PLA model in terms of $P(a_1), PP,$ and *TNR*. In the following experiments, we study the convergence rate of HLA and also its sensitivity to noise.

Experiment 6: Convergence rate

The ultimate objective of this experiment is to compare the convergence rate of the proposed model with that of FSLA and VSLA. To this aim, we are interested in the number of iterations needed for

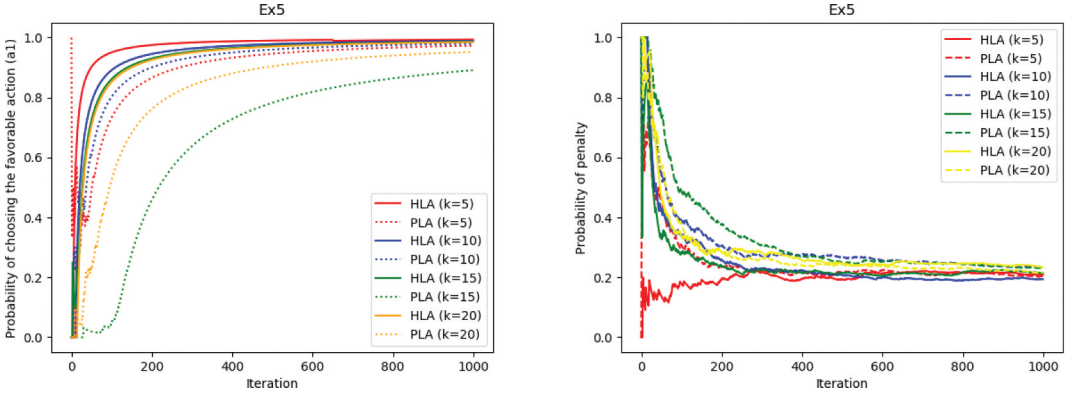

Figure 14. Experimental results of Ex5: the probability of choosing the favourable action, and probability of penalty in HLA and PLA with different number of allowable actions

Table 9. Experimental results of Ex5; Comparing HLA with PLA in terms of *TNR* and *TNAS*

Model	Ex5.1(k = 5)		Ex5.2(k = 10)		Ex5.3(k = 15)		Ex5.4(k = 20)	
	TNR	TNAS	TNR	TNAS	TNR	TNAS	TNR	TNAS
HLA	**818**	7	**784**	11	**798**	17	**761**	15
PLA	803	**3**	776	**1**	726	**5**	736	**2**

each LA to choose the favourable action in not less than $1 - \varepsilon$ fraction of times. Four different values of ε are considered: 0.01, 0.05, 0.1, and 0.2.

We used L_{R-I} as the updating schema with λ_1 and λ_2 being set to 0.1 and 0, respectively. The memory depth is also fixed with a value of 5. Four different cases for the number of allowable actions, or k, are considered: $k \in \{5, 10, 15, 20\}$. The probabilities of choosing each action are based on Table 2. The results of this experiment which are averaged over 100 runs, each with at least 10,000 iterations could be found in Figure 15.

Looking at the line graphs of Figure 15, one can observe that the higher the value of ε, the fewer iterations are needed for all LAs to reach a confidence level of $1 - \varepsilon$. Furthermore, as the number of allowable actions increases, it will be more challenging for FSLA and VSLA to choose the favourable action. The proposed model, however, works great even with large numbers of k.

Comparing the results of HLA with each model solely reveals many interesting points. In all cases, the proposed model exhibits a great improvement in comparison with VSLA. The reason is that the variable structure LA suffers from a lack of memory, while HLA uses an architecture similar to that of FSLA, and that would make this model inherit the benefits of FSLA. So, HLA is able to 'remember' the favourable action as soon as it finds that action for the first time. So, the expectation is that the proposed model must be able to converge sooner compared to VSLA (note a 25–50% speed up in the line graphs of Figure 15).

HLA, on the other hand, can perform as well as FSLA when high accuracy is not required for choosing the favourable action. When a small value of ε such as 0.01 is chosen, however, HLA outperforms fixed structure LA as well with a wide margin. The reason for this phenomenon is interesting. Imagine we chose 0.01 for ε. It means that we are looking for an iteration for which the LAs had reached a solid confidence level of 99% in terms of choosing the favourable action (α_1). FSLA could miss many iterations to find α_1 for the first time since there is no intelligence defined for this

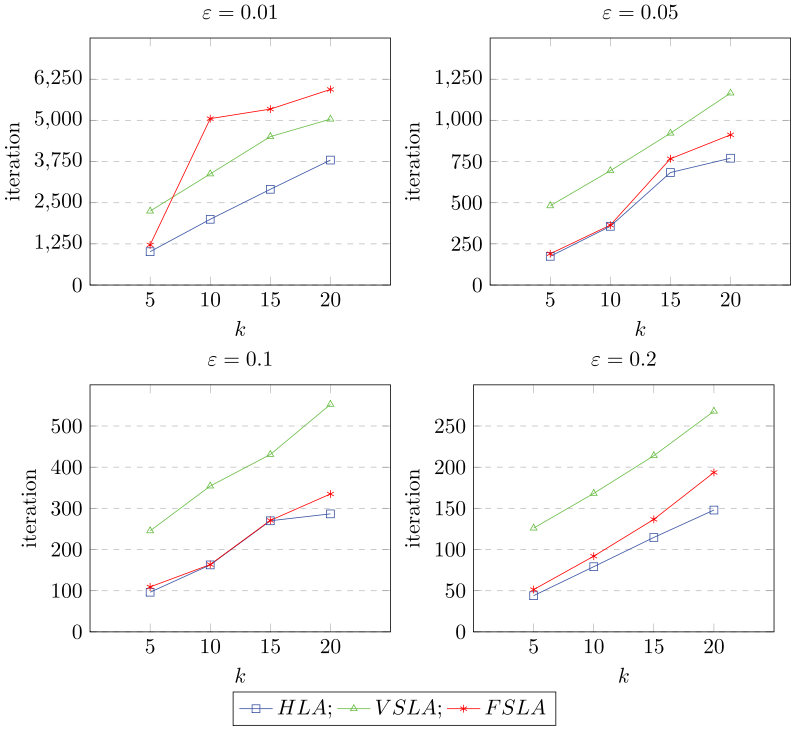

Figure 15. Experimental results of Ex6: Convergence rate of HLA, FSLA, and VSLA

model for choosing the next action. Thus, even if it can find α_1, it requires more iterations to reach that confidence level since it could have missed many action selection phases at the beginning. This is especially true for large values of k when finding the favourable action becomes exhausting for FSLA (look at the top-left graph in Figure 15 and also the similar concept for $k = 20$ in the left picture of Figure 13). But HLA performs impressively in similar conditions because of the embedded VSLA and its smart action selection. So, we do not expect our model to miss a large number of iterations at the beginning of finding α_1. This results in fewer iterations to achieve a higher degree of convergence for HLA compared to FSLA.

Consequently, based on the experimental results of Ex6 we argue that not only is HLA able to find the favourable action in just a few iterations, but it would also remember its choice for the next action selection phases. This would make the proposed model to have a significant improvement in terms of convergence compared to FSLA and VSLA.

Experiment 7: Sensitivity to noise

In this experiment, we aim to investigate the performance of LAs in noisy environments. So, instead of having a fixed reward probability for the favourable action (based on Table 2), this probability would be affected by a noise signal over time.

We used L_{R-I} as the updating schema with $\lambda_1 = 0.1$ and $\lambda_2 = 0$. Moreover, the memory depth is set to 5 and the LAs are to be chosen among $k = 20$ allowable actions. One of these actions is chosen to be the favourable action, randomly, and will have a higher chance of receiving a reward from the environment. The rests, however, are less likely for being rewarded. The reward probability of the favourable action is affected by a Gaussian noise signal in each iteration. The probability density function (PDF) p of a Gaussian random variable z is calculated based on Equation (7):

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{7}$$

Where σ is the standard deviation and μ is the mean. For instance, if $n = 100$ numbers are generated by Gaussian noise with $\mu = 0.5$ and $\sigma = 0.05$, we expect to see 100 numbers ranging mostly from 0.45 to 0.55, having a mean of 0.5. Therefore, for Ex7, we used a value of $\mu = 0.8$ so that the generated numbers will be close to 0.8 (the original reward probability). Changing σ from 0.01 to 0.2, with 0.01 increase in each experiment, will make the probability to have a greater noise in each experiment. No need to mention that Table 2 could be interpreted as a noise signal generated with $\mu = 0.8$ and $\sigma = 0$.

It should be noted that having above Gaussian noises, it is possible to see a number greater than 1 in some cases. Since the generated numbers stand for reward probability, we treat such cases as 1; meaning that the LAs will be rewarded when choosing the favourable action. Other actions, however, will have 0 chance of receiving a reward in that particular iterations. The performance of learning models will be judged based on their convergence. The value of ε is set to 0.01. Thus, we are looking for the number of iterations that are needed for each LA to select the favourable action (which has a noisy reward probability) with at least 99% chance. The results of this experiment are shown in Figure 16 and 17 and are averaged over 100 simulations.

As it is obvious in Figure 16, HLA is able to perform impressively in noisy environments, while VSLA and particularly FSLA seem to struggle in the same situation. To illustrate, HLA managed to reach a convergence of 99% in terms of choosing the favourable action in almost 4000 to 5000 iterations, even when the value of noise is greater than 0.1. VSLA, however, needs almost 5–6k iterations for reaching the same confidence level and the number of needed iterations for this model grows up significantly when $\sigma > 0.1$. On the other hand, FSLA is weaker when the number of allowable actions is high ($k = 20$) and also when the noise increases to more than 0.1, having a range of 8–10k of needed iterations in almost all experiments.

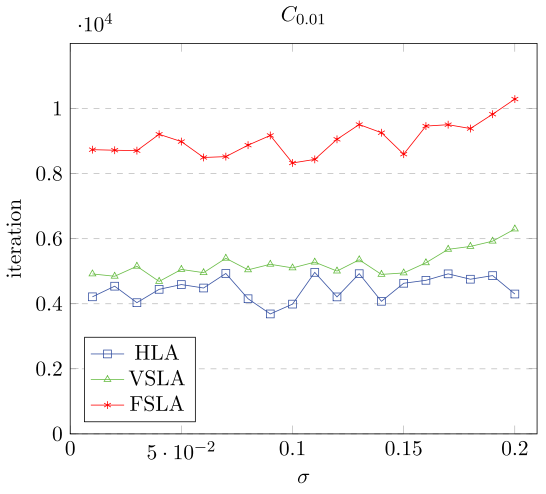

Figure 16. Experimental results of Ex7: Sensitivity to noise of HLA, FSLA, and VSLA in terms of convergence

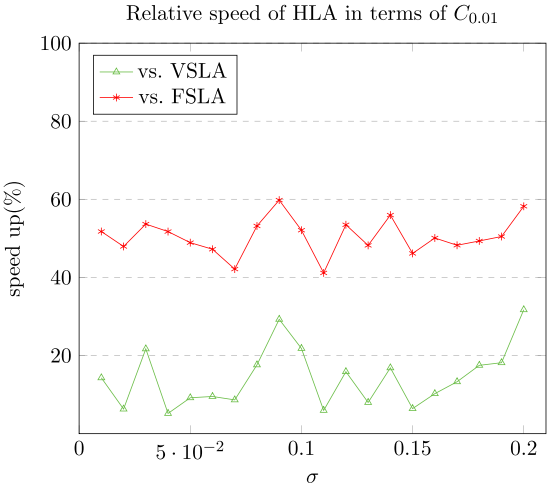

Figure 17. Experimental results of Ex7: Sensitivity to noise of HLA, FSLA, and VSLA in terms of convergence speed up

In Figure 17 the performance of HLA is compared to that of VSLA and FSLA in each experiment. For this comparison, we used $(C_{0.01}(VSLA) - C_{0.01}(HLA))/C_{0.01}(VSLA)$ and a similar equation for FSLA to calculate the relative speed up in each experiment. It is observed that HLA is able to perform 5–30% faster in comparison with VSLA particularly when the value of σ is greater than 0.15. The underlying reason is that since there is no memory defined for VSLA, this model might be affected heavily when it chooses the favourable action but will not receive a reward from the environment. Therefore, VSLA is likely to change its decision roughly in noisy situations. In the same scenarios, HLA will have great robustness and uses its memory to avoid abrupt action switching. Thus, it reaches a confidence level of $1 - \varepsilon$ sooner.

The speed up of HLA is even more promising compared to that of FSLA, having a range of 40–60% speed up in all experiments. This is due to the simple action switching mechanism of FSLA which chooses the next action, clockwise, when needed. It will take a while (at least k iteration in the best case) for this model to come back to the favourable action again. In contrast, HLA can manage to

Table 10. The setup of Ex8

	Ex8.1	Ex8.2	Ex8.3	Ex8.4	Ex8.5
k	3	3	3	3	3
N	0,1,2,3	0,1,2,3	0,1,2,3	0,1,2,3	0,1,2,3
λ_1	0.01	0.001	0.01	0	0
λ_2	0	0	0.01	0.01	0.1

choose the favourable action again even when an action switching is needed because of its embedded probability vector. This will help this learning model to perform better compared to FSLA.

In conclusion, the experimental results of Ex7 show the superiority of HLA to both VSLA and FSLA in noisy environments. Our proposed model can achieve a 30% speed up compared to VSLA and 60% in comparison with FSLA in terms of convergence.

Experiment 8: Application of HLA in Deep Neural Networks

This experiment is conducted to study the performance of the proposed HLA-based dropout algorithm. In this experiment, the GitHub code[2] is used as an implementation of the dropout mechanism. In order to develop the proposed HLA-based dropout algorithm, the function of *train_step*() given in their implementation is replaced with Algorithm *train_step*() that is explained in Figure 9. The parameters and configurations are explained as follows. The type of neural network is set to feedforward and the dataset is set to MNIST. We did not change other parameters in the code. Also, the input parameters of HLA are shown in Table 10. As can be seen, four different values for N are considered in each experiment: $N \in \{0, 1, 2, 3\}$. The numerical results of the simulation are shown in Table 11 in terms of mean accuracy and standard deviation.

In Table 11, the bold numbers represent the cases where our algorithm outperform the non-adaptive one. We need to mention that the mean accuracy and standard deviation for their algorithm are 0.9609 and 0.0007, respectively. Therefore, it is obvious that the proposed method can perform better compared to the non-adaptive model in many cases.

Also, as we discussed earlier (Section 4.1, remarks), an HLA with $N = 0$ could be considered as a normal VSLA. Thus, we can think of the first two columns of Table 11 as a novel solution for the same problem, using VSLA. Even in that case, the proposed solution could outperform the non-adaptive one (See Ex8.3, $N = 0$). However, once the value of N increases, the proposed algorithm seems to be more effective. For instance, in the case of $N = 3$, our algorithm performs better with a fairly good margin in Ex8.1 and Ex8.4.

We need to emphasise that the results of Table 11 also verify our claim regarding the robustness of HLA in terms of changing its parameters, since a good performance is observed in almost all experiments, regardless of the value of $N, \lambda_1,$ and λ_2. To sum up, the results of Experiment 8 illustrated that the fusion between VSLA and FSLA resulted in high performance in most cases.

Remarks

To sum up, in experiment 1–8, we compared the results of $HLA(k, N, \lambda_1, \lambda_2)$ with $FSLA(k, N)$ and $VSLA(k, \lambda_1, \lambda_2)$ under different circumstances. The proposed model also was compared to pursuit LA. All in all, the numerical results of our experiments reveal many interesting points, including:

Table 11. Experimental results of Ex8; Accuracy of neural networks with HLA-based dropout algorithm

	N = 0		N = 1		N = 2		N = 3	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Ex8.1	0.9585	0.0066	0.9586	0.0064	0.9573	0.0072	**0.9612**	**0.0060**
Ex8.2	0.9599	0.0064	0.9591	0.0065	0.9569	0.0073	0.9577	0.0069
Ex8.3	**0.9610**	**0.0032**	0.9600	0.0040	**0.9618**	**0.0037**	0.9591	0.0070
Ex8.4	0.9586	0.0052	**0.9612**	**0.0040**	0.9582	0.0061	**0.9619**	**0.0055**
Ex8.5	0.9594	0.0054	0.9581	0.0069	0.9604	0.0034	0.9535	0.0073

• In most cases, HLA outperforms both FSLA and VSLA in terms of more *TNR* and fewer *TNAS*. So, we claim that our proposed model has the benefits of both variable structure and fixed structure learning automata.

• Although VSLA is heavily dependent on the updating scheme, HLA is not affected by changing learning parameters λ_1 and λ_2. Therefore, even in P-model, when both λ_1 and λ_2 are set to 0, HLA performs impressively. The reason is that in such updating schemes, HLA tries to decide on the best action with its embedded FSLA.

• As opposed to FSLA which performs well with just high values of *N*, HLA achieves good results, whether *N* is high or not. The underlying reason is that the embedded VSLA of HLA learns to choose the best action regardless of the value of *N*. Consequently, as far as the HLA model is concerned, even if the automaton would need an action switching because of its low memory, there is a higher chance of choosing the favourableaction based on the VSLA unit.

• With higher values of allowable actions, *k*, the performance of VSLA and FSLA declines considerably due to the fact that the probability vector of each action would be extremely small. However, our proposed model, HLA, performs impressively even with a high value of *k*. This implies that our model is well suited even for the environments where many actions are allowed.

• HLA requires fewer *TNAS* in most cases since its probability vector learns how to choose the favorable action. Therefore, in cases when changing the action is costly, HLA would be compatible enough because of the fact that it avoids extra action switching.

• The robustness of HLA to changing each of the four parameters is surprisingly well. In other words, it performs well whether *N* is low or high, too many actions are allowed or not, and with any learning parameters. So, we conclude that the benefits of VSLA and FSLA are brought together in our proposed model.

• The penalty probability of HLA is considerably lower compared to that of VSLA and FSLA in all of the conducted experiments since the smart mechanism of action switching of HLA leads to a better action selection. This would make the proposed model to stick with the favorable action, and therefore, receive fewer penalties.

• HLA can outperform Pursuit LA since the proposed model has the benefits of both VSLA and FSLA. Consequently, instead of having a vector showing the number of times that each action has received a reward (as it is done in PLA), HLA remembers the chosen action by its embedded memory.

• The convergence rate of HLA is significantly quicker in comparison with fixed and variable structure LA. We predict a 25-50% improvement even when the number of allowable actions is high.

• HLA is expected to handle noise impressively, while the performance of FSLA and VSLA declines remarkably, particularly when the value of noise is too high. The results of our experiments demonstrate an improvement up to 60% in terms of convergence speed up.

• Lastly, HLA is able to perform well in real world problems, as we discussed in the last experiment. Our results show that HLA is able to improve the dropout mechanism in deep neural networks, and it illustrates that this model is applicable to many real-world applications.

Conclusion and Future Work

We have suggested a novel hybrid model, the HLA, based on both variable structure and fixed structure learning automata. The results of the computer simulations show it performs better than variable structure learning automata and fixed structure in most cases, as it inherits the learning capabilities of both models. In addition, the adaptive dropout mechanism based on the proposed model of learning automata performed better than the non-adaptive dropout mechanism as well as the dropout mechanism based on variable structure learning automata.

It should be noted that the proposed approach opens a new horizon for developing novel hybrid models of learning automata. It is obvious that more efforts should be invested in this field, and we cannot investigate all the possible combinations among VSLAs and FSLAs in this paper. Other

possible combinations should be considered in future work. The proposed approach could also be used to fuse two variable-structure learning automata. The proposed HLA design could also be used to organise a continuous action learning automaton that deploys multiple learning automata in its decision-making process.

Notes

1. https://github.com/wiseodd/hipsternet
2. https://github.com/wiseodd/hipsternet

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

Ali Mohammad Saghiri http://orcid.org/0000-0003-0797-314X

References

Aguilar, A. S. (1993). Learning Automata: An alternative to artificial neural networks. In *Neuroscience: From Neural Networks to Artificial Intelligence* (pp. 326–339). Springer.

Beigy, H., Meybodi, M. R., & Menhaj, M. B. (2002). Utilization of fixed structure learning automata for adaptation of learning rate in backpropagation algorithm. *Journal of Applied Sciences*, *2*(4), 437–443. https://scialert.net/abstract/?doi=jas.2002.437.443

Feng, S., Guo, H., Yang, J., Xu, Z., & Li, S. (2018). *A learning automata-based compression scheme for convolutional neural network*. In *International Conference in Communications, Signal Processing, and Systems* (pp. 42–49). Springer, Singapore.

Guo, H., Li, S., Li, B., Ma, Y., & Ren, X. (2017). A new learning automata-based pruning method to train deep neural networks. *IEEE Internet of Things Journal*, *5*(5), 3263–3269. https://doi.org/10.1109/JIOT.2017.2711426

Guo, H., Li, S., Qi, K., Guo, Y., & Xu, Z. (2018). *Learning automata based competition scheme to train deep neural networks*. *In IEEE Transactions on Emerging Topics in Computational Intelligence*, *4*(2), 151-158, https://doi.org/10.1109/TETCI.2018.2868474 .

Guo, H., Wang, S., Fan, J., & Li, S. (2019). Learning automata based incremental learning method for deep neural networks, *In IEEE Access*, *7*, 41164-41171. https://doi.org/10.1109/ACCESS.2019.2907645*Learning automata based incremental learning method for deep neural networks. IEEE Access* (Vol. 7, pp. 41164–41171).

Meybodi, M. R., & Beigy, H. (2002a). New learning automata based algorithms for adaptation of backpropagation algorithm parameters. *International Journal of Neural Systems*, *12*(01), 45–67. https://doi.org/10.1142/S012906570200090X

Meybodi, M. R., & Beigy, H. (2002b). A note on learning automata-based schemes for adaptation of BP parameters. *Neurocomputing*, *48*(1–4), 957–974. https://doi.org/10.1016/S0925-2312(01)00686-5

Narendra, K. S., & Thathachar, M. A. (2012). *Learning automata: An introduction*. Courier corporation.

Rezvanian, A., Saghiri, A. M., Vahidipour, S. M., Esnaashari, M., & Meybodi, M. R. (2018). *Recent advances in learning automata* (Vol. 754). Springer International Publishing.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

Sudareshan, M. K., & Condarcure, T. A. (1998). Recurrent neural-network training by a learning automaton approach for trajectory learning and control system design. *IEEE Transactions on Neural Networks*, *9*(3), 354–368. https://doi.org/10.1109/72.668879

Thathachar, M. A., & Sastry, P. S. (2011). *Networks of learning automata: Techniques for online stochastic optimization*. Springer Science and Business Media.

Yazidi, A., Oommen, B. J., & Goodwin, M. (2016). On solving the problem of identifying unreliable sensors without a knowledge of the ground truth: The case of stochastic environments. *IEEE Transactions on Cybernetics*, *47*(7), 1604–1617. https://doi.org/10.1109/TCYB.2016.2552979

Yazidi, A., Zhang, X., Jiao, L., & Oommen, B. J. (2019). The hierarchical continuous pursuit learning automation: A novel scheme for environments with large numbers of actions. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(2), 512–526. https://doi.org/10.1109/TNNLS.2019.2905162

Zhang, X., Jiao, L., Oommen, B. J., & Granmo, O. C. (2019). A conclusive analysis of the finite-time behavior of the discretized pursuit learning automaton. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(1), 284–294. https://doi.org/10.1109/TNNLS.2019.2900639